

TP C#0 : There and Back Again

Consignes de rendu

A la fin de ce TP, vous devrez rendre une archive respectant l'architecture suivante :

```
rendu-tp0-prenom.nom.zip
|-- prenom.nom/
|   |-- AUTHORS
|   |-- README
|   |-- TP0/
|       |-- TP0.sln
|       |-- TP0/
|           |-- Everything except bin/ and obj/
```

N'oubliez pas de vérifier les points suivants avant de rendre :

- Remplacez `prenom.nom` par votre propre login et n'oubliez pas le fichier `AUTHORS`.
- Les fichiers `AUTHORS` et `README` sont obligatoires.
- Pas de dossiers `bin` ou `obj` dans le projet.
- Respectez scrupuleusement les prototypes demandés.
- Retirez tous les tests de votre code.
- **Le code doit compiler !**

AUTHORS

Ce fichier doit contenir une ligne formatée comme il suit : une étoile (*), un espace, votre login et un retour à la ligne. Voici un exemple (où \$ est un retour à la ligne et _ un espace) :

```
*_firstname.lastname$
```

Notez que le nom du fichier est `AUTHORS` sans 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, vous devez lister et expliquer tous les boni que vous aurez implémentés. Un `README` vide sera considéré comme une archive invalide (malus).

1 Introduction

1.1 Objectifs

Durant ce TP, vous allez découvrir et apprendre à utiliser un langage relativement nouveau. Nous parlons ici du **C#**, qui est développé et maintenu par une grande multinationale : Microsoft. Vous travaillerez également sur un tout nouvel environnement : **Rider** par JetBrains (on salue Emacs en partant !).

1.2 Le langage C#

Le langage C# (prononcé "Si sharp") est un langage de programmation orienté objet développé et sponsorisé par Microsoft. Originellement supporté par Windows, le langage C# est désormais disponible sur Linux et MacOS. Ce changement a eu lieu majoritairement à l'initiative de Mono.

Dérivé comme beaucoup des langages C et C++, le C# est en grande partie inspiré de Java, dont il emprunte certains concepts et idées. Si certains mots ou langages vous sont inconnus, nous vous invitons à lire leur page Wikipédia. Vous les rencontrerez très certainement pendant vos trois premières années à EPITA. Il n'est jamais trop tôt pour s'informer !

Pour en revenir au C#, ce langage est **très** différent du langage OCaml que vous avez appris à adorer. Pour le meilleur ou pour le pire. Dans tous les cas, ce document est un survol rapide des bases du langage. Nous vous recommandons d'aller voir la documentation officielle **mdsn.com** pour plus de détails sur les spécificités du langage, et une liste exhaustive des fonctions et bibliothèques disponibles.

1.3 Rider

Rider, par JetBrains, est une plateforme IDE (Integrated development environment) pour le langage C#. Si vous avez déjà codé dans un langage un tant soit peu populaire, il est possible que vous ayez déjà utilisé un IDE de JetBrains : IntelliJ IDEA pour Java, PyCharm pour Python, WebStorm pour JavaScript, *etc.*

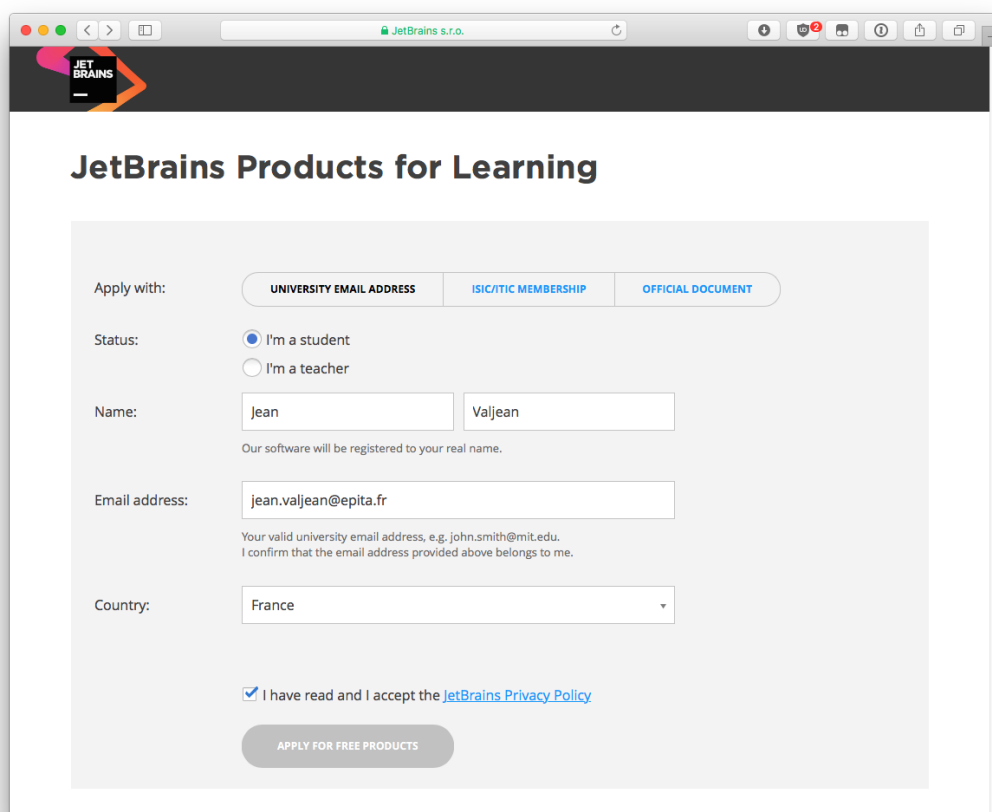
Comme tous les IDEs modernes, Rider fournit un système d'auto-complétion intelligent, un débogueur puissant et de nombreux outils plus ou moins utiles. Nous ne les aborderons bien sûr pas tous cette année, c'est pourquoi nous vous recommandons d'aller explorer les capacités de Rider de votre côté ! Savoir l'utiliser vous économisera beaucoup de temps de développement.

2 On apprend à surfer

2.1 La licence Rider

Rider est un logiciel payant, et vous devrez utiliser une licence pour pouvoir l'utiliser sur la durée. Heureusement, en tant qu'étudiants, c'est gratuit !

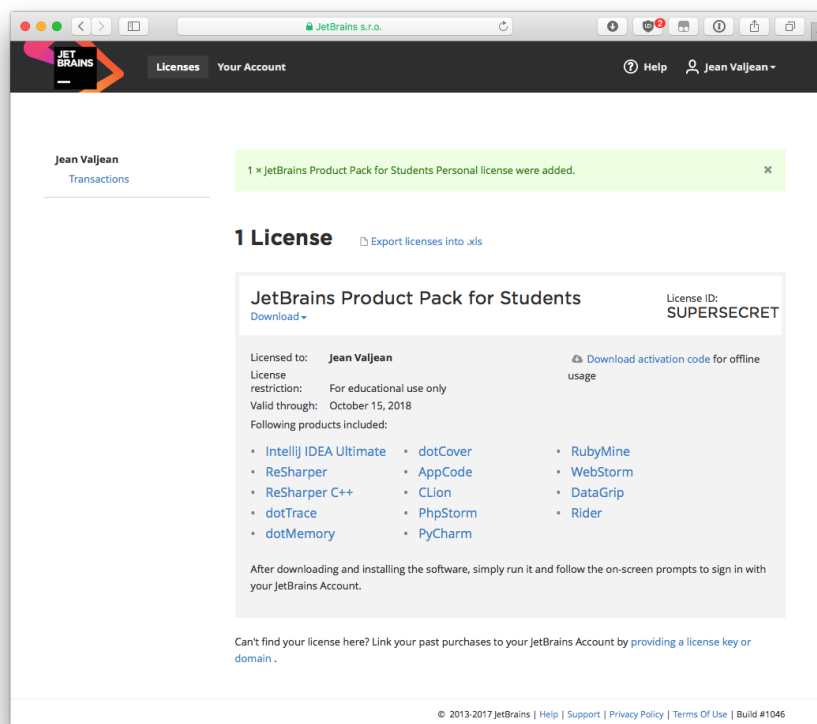
Allez à l'URL <https://www.jetbrains.com/shop/eform/students> et entrez votre nom, prénom et **adresse mail EPITA**, puis cliquez sur "Apply". Le formulaire devrait ressembler à ça, avec vos informations :



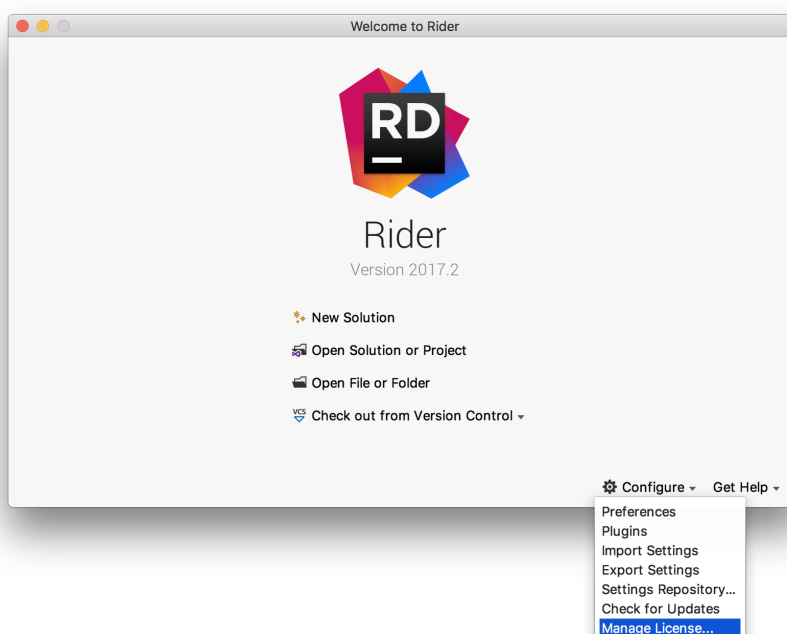
The screenshot shows a web browser window displaying the "JetBrains Products for Learning" registration form. The form is titled "JetBrains Products for Learning" and includes the following fields and options:

- Apply with:** Three tabs: "UNIVERSITY EMAIL ADDRESS" (selected), "ISIC/TIC MEMBERSHIP", and "OFFICIAL DOCUMENT".
- Status:** Two radio buttons: "I'm a student" (selected) and "I'm a teacher".
- Name:** Two text input fields. The first field contains "Jean" and the second field contains "Valjean". Below the fields, it says: "Our software will be registered to your real name."
- Email address:** A text input field containing "jean.valjean@epita.fr". Below the field, it says: "Your valid university email address, e.g. john.smith@mit.edu. I confirm that the email address provided above belongs to me."
- Country:** A dropdown menu with "France" selected.
- Privacy Policy:** A checkbox labeled "I have read and I accept the [JetBrains Privacy Policy](#)" is checked.
- Submit Button:** A button labeled "APPLY FOR FREE PRODUCTS".

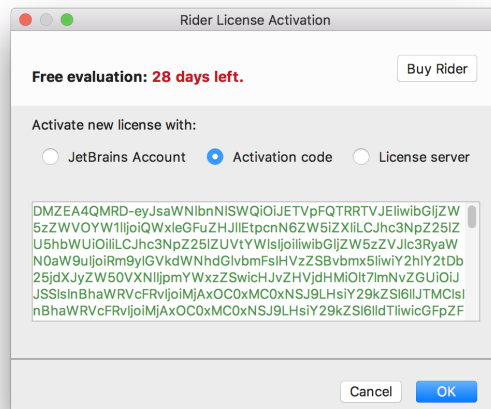
Lorsque que vous aurez envoyé le formulaire et validé la demande de licence reçue par mail, vous pourrez créer un compte, avec votre adresse e-mail préremplie et un mot de passe long et sécurisé. Votre écran devrait ensuite ressembler à ceci :



Cliquez sur le lien "Download activation code", ouvrez le document téléchargé et copiez son contenu. Puis, ouvrez Rider. Si l'écran de licence ne s'ouvre pas, sur l'écran d'accueil, dirigez-vous vers "Configure" et choisissez "Manage License..."



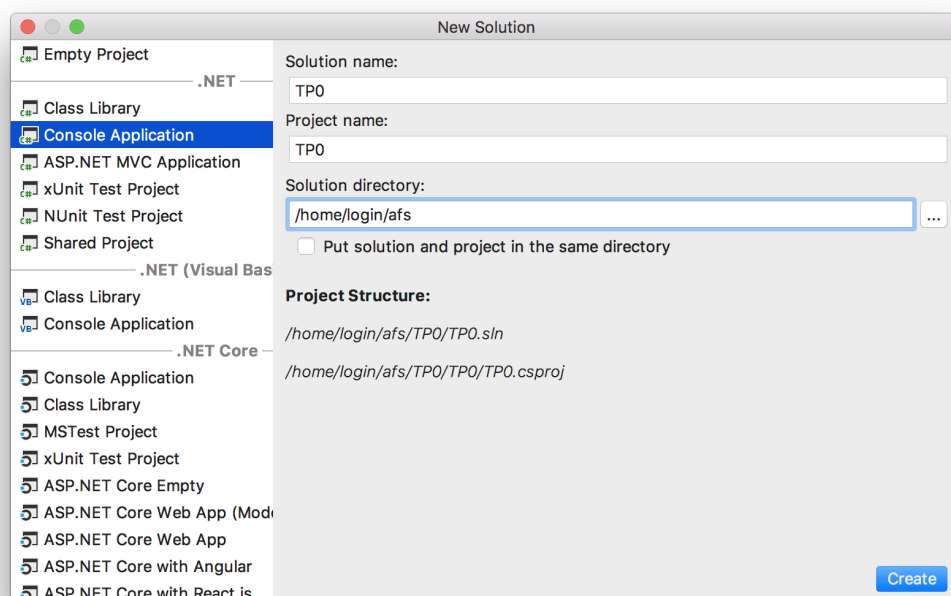
Copiez votre code d'activation (qui correspond - quel miracle! - à ce que vous avez copié deux étapes au dessus).



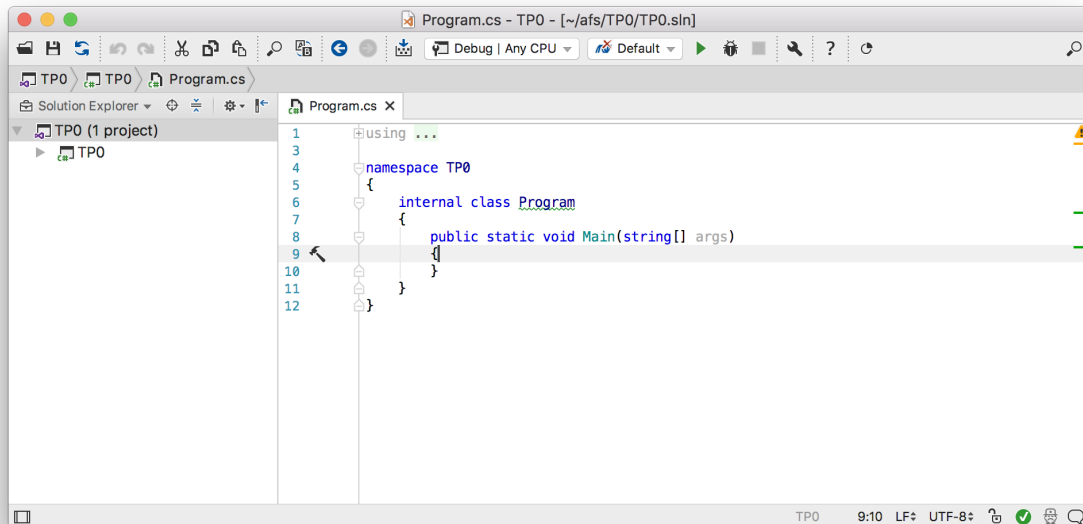
Félicitations, voilà une bonne chose de faite! Il est temps de passer aux choses pratiques...

2.2 Créer un nouveau projet

Sur l'écran d'accueil, cliquez sur le lien "New Solution" et choisissez les paramètres suivants :



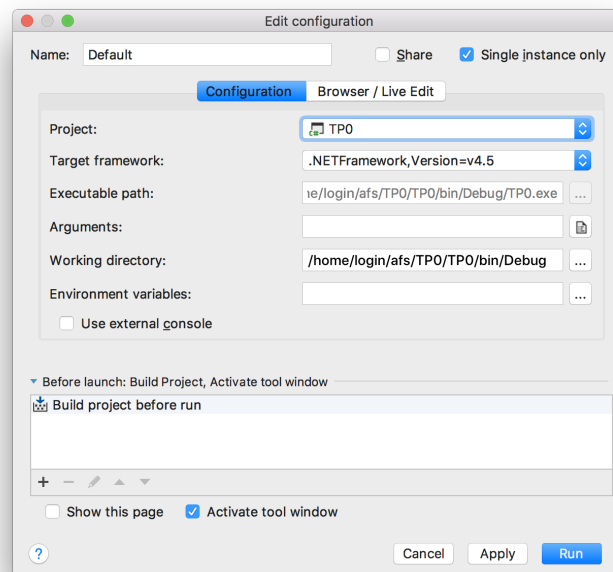
Bien évidemment, nous vous demandons de remplacer "login" par votre propre login. Cliquez maintenant sur "Create". Vous serez accueillis par cet écran :



À gauche, vous pouvez voir la hiérarchie de votre projet. Cliquer sur les petites flèches vous permettra d'agrandir et de montrer les fichiers et dossiers imbriqués dans votre projet. Pour l'instant, le seul fichier qui nous intéresse est "Program.cs" : c'est là que vous écrirez votre code pour ce TP.

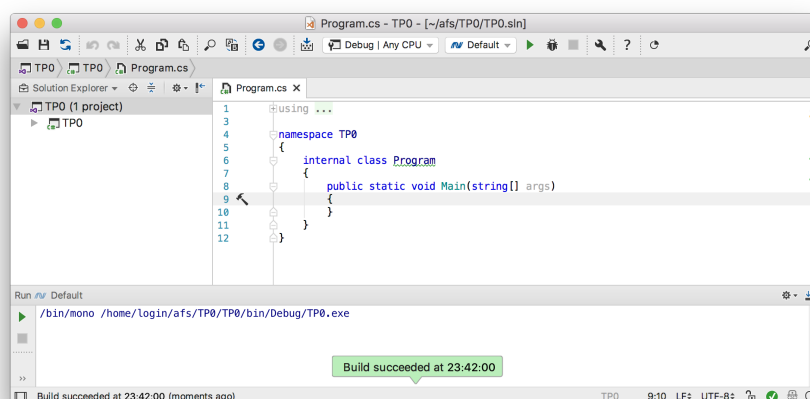
À droite, vous trouverez l'éditeur de texte. Par défaut, le fichier "Program.cs" est déjà ouvert.

En haut, vous trouverez la boîte à outils. Pour l'instant, le seul bouton qui nous intéresse est la petite flèche verte pointant vers la droite. Elle vous permettra de compiler et de lancer votre code. En passant votre souris dessus, vous verrez également le raccourci clavier correspondant. Cliquez dessus.



Sur l'image ci-dessus, vous pouvez voir les options par défaut de compilation et d'exécution de votre programme. Vous n'avez rien besoin de changer, alors cliquez simplement sur "Run". La prochaine fois que vous lancerez votre programme, cet écran n'apparaîtra plus.

Enfin, nous arrivons à l'onglet en bas de votre écran, avec la commande d'exécution de votre programme et la sortie console de votre programme (qui n'affiche rien pour l'instant). Cet onglet est appelé la *console*, et vous apprendrez à l'utiliser lors de ce TP ainsi que du prochain.



Et maintenant... Place à la programmation !

3 C# 101

Les notions que nous allons aborder sont les bases du langage. Vous aurez l'occasion de les revoir dans les prochains TPs, lorsque nous aborderons des principes plus avancés du C# .

Une fois encore, si une notion vous échappe ou que vous souhaiteriez explorer un sujet plus en détails, nous vous invitons à aller voir la documentation officielle du langage sur **mdsn.com**. Ce document n'est qu'une pâle introduction, où nous présumons que vous connaissez déjà les bases d'un langage de programmation.

En C# , comme dans la plupart des langages basés sur le C :

- Votre **programme** est une séquence d'**instructions**.
- Une **instruction** se termine toujours pas un point-virgule (;).
- Les **variables** sont définies avec un **type** et se voient associer une valeur.
- Une **fonction** est un groupe d'instructions qui prennent des **arguments** et peuvent **renvoyer** une valeur.

Voici un exemple de programme qui affiche "chatons" quand il est exécuté :

```
1 using System;
2
3 namespace TP0
4 {
5     internal class Program
6     {
7         public static void Main(string[] args)
8         {
9             Console.WriteLine("chatons");
10        }
11    }
12 }
```

Heureusement, la plupart des programmes que nous vous demanderons d'écrire seront un poil plus complexe.

Concentrons-nous surtout sur le bout de code qui nous intéresse :

```
1 public static void Main(string[] args)
2 {
3     Console.WriteLine("chatons");
4 }
```

Cette partie définit une fonction appelée **Main**, qui prend une liste de chaînes de caractères et ne retourne rien (**void**). Vous pouvez oublier les mots-clés **public** et **static** pour l'instant, vous les reverrez lors d'un prochain TP.

Quant au bout de code **Console.WriteLine**, comme vous l'avez probablement deviné, il s'agit d'une fonction qui prend une chaîne de caractères et l'affiche dans la console.

Durant ce TP, nous vous demanderons d'écrire des fonctions aux prototypes et aux comportements variés. Les prototypes vous seront donnés. Nous vous demandons de les respecter, afin que votre travail puisse être noté.

Voici un exemple d'écriture et de test de vos fonctions dans votre fichier `Program.cs` :

```
1  using System;
2
3  namespace TP0
4  {
5      internal class Program
6      {
7          public static void Main(string[] args)
8          {
9              Console.WriteLine(Test(2022));
10         }
11
12         public static int TheAnswer()
13         {
14             return 42;
15         }
16
17         public static bool Test(int answer)
18         {
19             return answer == TheAnswer();
20         }
21     }
22 }
```

La fonction `Main` sera appelée au début de votre programme. Il s'agit donc de l'endroit parfait pour tester vos fonctions !

3.1 Déclarer des variables

La syntaxe de déclaration de variables est la suivante :

```
1  int yourGrade;    // Déclaration
2  yourGrade = -42;  // Instruction d'affectation
3  yourGrade = 0;    // Nouvelle instruction d'affectation
4  float pi = 3.14;  // Déclaration + Affectation
```

Ici, les mots-clés `int` et `float` représentent le **type** de vos variables, qui sont respectivement un nombre entier signé et un nombre à virgule flottante. En OCaml, vous n'aviez pas à vous occuper du type. Vous ne pouviez également pas déclarer de variable sans lui affecter une valeur, ni changer sa valeur sans la redéclarer. Hélas, le bon sens est relatif.

Toutefois, vous pouvez laisser C# décider du type en utilisant le mot-clé `var`.

```
1  var hey = "that's pretty good"; // `hey` est une chaîne de caractères
```

Attention ! Les variables déclarés avec le mot-clé `var` ont toujours un type : il n'est pas possible de changer le type d'une variable en la réaffectant. Il est également impossible de déclarer une variable avec `var` sans l'initialiser avec une valeur.

Comme exemple, les lignes suivantes déclencheront une erreur du compilateur :

```
1 var what; // Error: Implicitly-typed local variable must be initialized.
2 var pi = 3.14;
3 pi = "3.14";
4 // Error: Cannot convert source type `string` to target type `double`.
```

Voici une liste des types les plus courants que vous trouverez en C# :

Identifiant	Exemple	Description
bool	true	Un de ces deux états : true (vrai) ou false (faux)
byte	42	Un nombre entier de 0 à 255
short	420	Un nombre entier de -2^{15} à $2^{15} - 1$
int	-50000	Un nombre entier de -2^{31} à $2^{31} - 1$
long	666L	Un nombre entier de -2^{63} à $2^{62} - 1$
ushort	420	Un nombre entier de 0 à $2^{16} - 1$
uint	50000	Un nombre entier de 0 à $2^{32} - 1$
ulong	666L	Un nombre entier de 0 à $2^{64} - 1$
float	4.2f	Une nombre à précision simple de -3,402823e38 à 3,402823e38
double	4.2d or 4.2	Un nombre à précision double de -1,79769313486232e308 à 1,79769313486232e308
char	'a'	Un caractère
string	"epita"	Une séquence de zéro ou plus caractères

3.1.1 Opérateurs arithmétiques

Voici une liste des opérateurs arithmétiques les plus courants que vous serez amenés à utiliser en C# :

Usage	Signification
-a	La négation de a
a + b	L'addition de a et b
a - b	La soustraction de b à a
a * b	La multiplication de a et b
a / b	La division de a par b
a % b	Le reste de la division de a par b (Modulo)

3.1.2 Chaînes : propriétés, méthodes et opérateurs

Dans ce TP, vous aurez besoin d'utiliser les opérations suivantes sur les chaînes :

Usage	Résultat
s.Length	Taille de la chaîne s
s.Substring(n)	Copie de la chaîne s commençant à l'index n
s[n]	Le caractère à l'index n dans la chaîne s

3.2 Les conditions

3.2.1 Les conditions if/else/else if

Comme vous vous en doutez, le C# déclare sa propre syntaxe pour les constructions if et else :

```
1  var cool = "chat";
2  if (cool == "chat")
3  {
4      // Quelque chose d'intéressant
5  }
6  else
7  {
8      // Quelque chose d'autre
9  }
```

Le if comme le else doivent être suivis soit d'une instruction unique, soit d'un bloc d'instructions. Un bloc d'instructions est défini comme une séquence d'instructions et est entouré d'accolades ("{" and "}"). Il est cependant possible d'omettre les accolades si il n'y a qu'une seule instruction dans votre bloc.

De ce fait, ces deux bouts de code sont équivalents :

```
1  if (1 + 1 == 2)
2  {
3      Console.WriteLine("Sounds good");
4  }
5  else
6  {
7      Console.WriteLine("Doesn't look like anything to me");
8  }
```

```
1  if (1 + 1 == 2)
2      Console.WriteLine("Sounds good");
3  else
4      Console.WriteLine("Doesn't look like anything to me");
```

Une conséquence de cette définition est que vous pouvez utiliser plusieurs conditions à la suite sans devoir les imbriquer les unes dans les autres :

```
1  if (1 + 1 == 2)
2  {
3      // Do something important.
4  }
5  else
6  {
7      if (2 + 2 == 3)
8      {
9          // Where did we go wrong?
10     }
11 }
12 // Devient
13 if (1 + 1 == 2)
14 {
15     // Do something important.
16 }
17 else if (2 + 2 == 3)
18 {
19     // Where did we go wrong?
20 }
```

Note : `cool == "cat"` est appelé une expression booléenne, car il s'agit d'une expression qui se résout en valeur booléenne (en C# , soit `true`, soit `false`). La condition `if` attendra **toujours** une valeur booléenne.

Attention ! Les opérateurs booléens en C# sont un peu différents de ceux que vous avez utilisés en OCaml. Nous vous dirigeons vers **la documentation officielle** pour une liste exhaustive.

3.2.2 Opérateurs de comparaison

Voici une liste des opérateurs les plus communs que vous rencontrerez en C# sur les conditions :

Usage	Signification
<code>a == b</code>	a est égal à b
<code>a != b</code>	a est différent de b
<code>a < b</code>	a est plus petit que b
<code>a > b</code>	a est plus grand que b
<code>a <= b</code>	a est plus petit que ou égal à b
<code>a >= b</code>	a est plus grand que ou égal à b

3.2.3 Opérateurs logiques

Vous pouvez combiner plusieurs expressions booléennes à l'aide d'opérateurs logiques. Dans le tableau ci-dessous, A et B sont des expressions booléennes.

Usage	Signification
A && B	A ET B
A B	A OU B
!A	NON A

Tout comme en OCaml, ces opérateurs sont *lazy* (séquentiels) : ils vont comparer de gauche à droite et s'arrêter dès que la valeur de l'opération globale peut être déterminée. Par exemple, dans l'exemple suivant :

```
1 bool isCat = (IsAnimal(thing) && HasWhiskers(thing)) || Meows(thing);
```

Quand `IsAnimal(thing)` renvoie `false`, `HasWhiskers(thing)` n'est jamais évalué, puisque `false && qqchose` sera toujours évalué comme `false`. De même, si `IsAnimal(thing) && HasWhiskers(thing)` renvoie `true`, `Meows(thing)` ne sera pas évalué puisque `true || qqchose` renvoie toujours `true`.

3.2.4 Opérateur ternaire

Il arrive que l'on veuille utiliser des conditions à l'intérieur d'expressions. Malheureusement, en C# , les conditions `if/else` sont des instructions (*statements*) et non des **expressions**. C'est pourquoi nous utilisons l'**opérateur ternaire** :

```
1 int sign = x < 0 ? -1 : 1;
```

Voyez-le comme une question : Est-ce que x est inférieur à 0 ? Si oui, utilise -1, sinon utilise 1.

Attention ! N'abusez pas de l'opérateur ternaire. Bien qu'il soit possible de les imbriquer, vous devriez toujours privilégier un code lisible à un code court. Souvenez-vous que la majorité du temps en programmation est consacré à la lecture du code et non à son écriture !

3.2.5 L'instruction switch

Vous vous souvenez du pattern matching? Vous allez rencontrer son lointain cousin. Le `switch` vous permet de tester une expression sur plusieurs valeurs prédéfinies et d'exécuter un bloc de code en conséquence. Sa syntaxe est la suivante :

```
1  switch (EXPRESSION0)
2  {
3  case EXPRESSION1:
4      // Do something meaningful
5      break;
6  case EXPRESSION2:
7      // Do something for the greater good
8      break;
9  default:
10     // Do something else I guess
11     break;
12 }
```

Ici, `EXPRESSION{0,1,2}` peuvent représenter n'importe quelles expressions. La branche `default` est optionnelle, et sera visitée uniquement si aucun cas n'a pu être associé.

Par exemple, dans le code suivant :

```
1  switch (40 + 2)
2  {
3  case 7 * 6:
4      Console.WriteLine("What a great coincidence");
5      break;
6  case 20 * 2:
7      Console.WriteLine("Close enough");
8      break;
9  default:
10     Console.WriteLine("Who needs maths anyway");
11     break;
12 }
```

Seule la première branche sera visitée et `What a great coincidence` sera affiché à l'écran.

Attention ! N'oubliez pas d'ajouter la ligne `break`; à la fin de vos cas. Sinon, le programme sautera directement au cas suivant, même si le cas ne peut être associé à la valeur donnée. Voici un exercice pour le lecteur : qu'affiche le code suivant pour `count` allant de 0 à 6 ?

```
1  switch (count)
2  {
3      case 0:
4      case 1:
5      case 2:
6          Console.WriteLine("Too few kittens");
7          break;
8      case 3:
9      case 4:
10         Console.WriteLine("An acceptable amount of kittens");
11         break;
12     default:
13         Console.WriteLine("An unorthodox number of kittens");
14         break;
15 }
```

3.3 Quelques outils supplémentaires...

Pour finir ce TP, vous aurez besoin de quelques méthodes et attributs que nous n'avons pas encore mentionnés dans ce sujet. À savoir :

- `Console.ReadLine()` : lit une ligne d'entrée de l'utilisateur dans la console et la retourne comme `string`.
- `int.Parse(string)` : prend une `string` et la convertit en une valeur `int`.
- `DateTime.Today` : une structure de donnée qui correspond à la date d'aujourd'hui. Essayez d'écrire `DateTime.Today` dans votre éditeur, sans oublier le point à la fin : l'éditeur va automatiquement essayer de compléter votre expression avec les attributs qui pourraient vous intéresser !

4 Exercices

Les exercices suivants utilisent uniquement le cours de C# que vous venez de terminer. Souvenez-vous de la récursion ! Vous en aurez besoin.

Faites attention à bien gérer tous les cas ! (notamment les nombres négatifs)

4.1 Hello World

Et on recommence par la base ! Il vous faut afficher **Hello World!** sur la console.

```
1 static void HelloWorld();
```

4.2 Say Hello

Dire bonjour, c'est bien. Avec un prénom, c'est mieux. Il vous faut maintenant demander sur la console le prénom de l'utilisateur, et le saluer.

Exemple, où les lignes paires représentent les entrées utilisateur :

```
1 What's your name?  
2 Alex  
3 Well hello Alex!
```

```
1 static void SayHello();
```

4.3 Calculer un âge

Vous devez demander l'année de naissance de l'utilisateur et afficher son âge approximatif.

Exemple, où les lignes paires représentent les entrées utilisateur :

```
1 What's your year of birth?  
2 1999  
3 Looks like you're around 18!
```

```
1 static void CalcAge();
```

4.4 Valeur absolue

La fonction **Absolute** renvoie la valeur absolue de l'entier **x**.

```
1 static int Absolute(int x);
```

4.5 Factorielle

La fonction **MyFact** renvoie la factorielle de l'entier non signé **n**.

```
1 static uint MyFact(uint n);
```


4.6 Puissance

La fonction `MyPow` renvoie le nombre `x` à la puissance `n`. **Attention !** Il faut gérer les puissances négatives.

```
1 static double MyPow(double x, int n);
```

4.7 Fibonacci

La fonction `MyFibo` a comme valeur de retour :

- $\text{MyFibo}(N) = \text{Fibo}(n-1) + \text{Fibo}(n-2)$
- $\text{MyFibo}(1) = 1$
- $\text{MyFibo}(0) = 0$

```
1 static uint MyFibo(uint n);
```

4.7.1 Édition de String

La fonction `ChangeChar` prend une chaîne de caractères `s`, un caractère `c` et un entier positif `n`, et renvoie la string `s` modifiée avec `c` à la `nth` place.

Exemple :

- $\text{ChangeChar}(\text{"hello"}, 'a', 1) = \text{"hallo"}$
- $\text{ChangeChar}(\text{"hello"}, 'n', 6) = \text{"hello"}$

```
1 static string ChangeChar(string s, char c, uint n);
```

4.7.2 PGCD

La fonction `MyGcd` renvoie le plus grand diviseur commun de deux nombres entiers donnés en paramètres (*Greatest Common Divisor*).

```
1 static uint MyGcd(uint a, uint b);
```

4.8 BONUS

4.8.1 Racine carrée

La fonction `MySqrt` renvoie l'approximation de la racine carrée d'un nombre positif `n` sur `i` itérations en utilisant la méthode de Héron. La version récursive de l'algorithme donne :

- $r_0 = \frac{1}{2}$
- $r_{i+1} = \frac{r_i + \frac{n}{r_i}}{2}$

```
1 static double MySqrt(double n, uint i);
```

4.8.2 Inversion de String

La fonction `MyReverseString` prend une chaîne de caractères et l'inverse. Évidemment, interdiction d'utiliser `.Reverse()` : les seuls opérateurs autorisés sont décrits subsection 3.1.2! Exemple : `MyReverseString("hello") == "olleh"`

```
1 static string MyReverseString(string s);
```

4.8.3 Palindrome

La fonction `MyIsPalindrome` prend une chaîne de caractères de type **string** et renvoie **true** si il s'agit d'un palindrome, **false** sinon. Encore une fois, interdiction d'utiliser d'autres méthodes ou opérateurs que ceux décrits dans ce document !

Exemple :

```
— MyIsPalindrome("hello olleh") == true  
— MyIsPalindrome("kayak") == true  
— MyIsPalindrome("Kayak") == false
```

```
1 static bool MyIsPalindrome(string a);
```

4.9 Calculer l'âge réel !

Maintenant vous devez demander la date de naissance de l'utilisateur et afficher son âge exact !

Exemple :

```
1 What's your year of birth?  
2 1999  
3 What's your month of birth?  
4 12  
5 What's your day of birth?  
6 11  
7 Looks like you're exactly 17!
```

```
1 static void CalcRealAge();
```

These violent deadlines have violent ends.