Chapitre 4. .NET 3.0/3.5/4.*/5.0 (nouveautés du langage), LINQ

> "L’informatique semble encore chercher la recette miracle qui permettra aux gens d’écrire des programmes corrects sans avoir à réfléchir. Au lieu de cela, nous devons apprendre aux gens comment réfléchir."
>
> —Anonyme

Séance 4/10 du 01/02/2021, d'une durée de 4h

Objectifs de la séance

Compétences	**Objectifs**
La plateforme .NET en 2021	Connaître les dernières possibilités de .NET 3.5, 4.*, 5.0
Simplifications de base	Apprendre les types implicites et les propriétés auto-implémentées.
Les initialiseurs	Déclarer et initialiser facilement des tableaux, objets, ...
Les anonymes	Connaître les types et méthodes anonymes
Les expressions lambda	Découvrir une simplification des opérations de calculs ou de recherches
Les extensions	Comment faire pour rajouter des méthodes à des classes existantes non accessibles
Appel des méthodes dynamique	Comment appeler des méthodes non disponibles lors de la compilation ?
Paramètre nommé et optionnel	Comment pouvoir passer des paramètres dans le désordre ou partiellement ?
Nouveautés de C# avec .NET 4.5	Codage des procédures asynchrones et de leur appel.
Nouveautés du C# 6	Quelques mises en pratique des dernières nouveautés du langage avec Visual Studio
Nouveautés du C# 7	
Nouveautés du C# 8	
Nouveautés du C# 9	
Linq	Découvrir la puissance du langage de requète intégré de la plateforme .NET
Travaux dirigés	Mise en pratique des techniques apprises
Master.HelpDesk	Contrôle des mise à jour, compilation, contrôle du style et des commentaires.
Découverte d'un nouvel outil de développement	Découverte de la documentation des sources, Ghostdoc, génération automatique de la documentation technique avec SandCastle.

1. La plateforme .NET en 2021

Déroulement

Phase	Durée	Type de travail	Modalités de travail
La plateforme .NET en 2021	60 mn	Cours, travaux dirigés	Individuel ou par binome
Connaître les dernières possibilités de .NET 3.5, 4.*, 5.0			

Après la présentation des dernières versions en cours de Visual Studio et des plateformes et technologies déployées, nous aborderons plus en détail les nouveautés du langage, en particulier Linq.

1.1. Visual Studio 2008 et .NET 3.5 SP1

Les dernières versions de la plateforme (3.*) et de Visual Studio vont modifier notre manière de développer grâce à des simplifications ou des possibilités d'extensions ou de manipulations :

LINQ (Language Integrated Query) est une nouvelle fonctionnalité de Visual Studio 2008 et du .NET Framework 3.5. LINQ étend des fonctions de requête puissantes dans la syntaxe de langage de C# et Visual Basic, sous la forme de modèles de requête standard et facilement assimilables. Cette technologie peut être étendue pour prendre en charge potentiellement tout type de magasin de données. Le .NET Framework 3.5 inclut des assemblys de fournisseur LINQ qui vous permettent d'utiliser LINQ pour interroger des collections .NET Framework, des bases de données SQL Server, des groupes de données ADO.NET et des documents XML.

Accès aux données avec ADO.NET : Le fournisseur de données .NET Framework pour SQL Server (System.Data.SqlClient) fournit la prise en charge complète de toutes les nouvelles fonctionnalités du moteur de base de données SQL Server 2008. Pour plus d'informations sur la prise en charge .NET Framework pour SQL Server 2008, consultez Nouvelles fonctionnalités de SQL Server 2008 (ADO.NET). La plateforme de données ADO.NET est une stratégie multi-version permettant de diminuer la quantité de codage et la maintenance nécessaires pour les développeurs en leur permettant de programmer par rapport à des modèles EDM (Entity Data Model) conceptuels. Cette plateforme inclut Entity Framework ADO.NET, le modèle EDM (Entity Data Model), Object Services, LINQ to Entities, Entity SQL, EntityClient, les Services de données ADO.NET et les outils EDM. Pour plus d'informations, consultez Plateforme de données ADO.NET.

Intégration ASP.NET AJAX et WCF : L'intégration de WCF avec les fonctionnalités JavaScript et XML asynchrones (AJAX) dans ASP.NET fournit un modèle de programmation de bout en bout pour la génération des applications Web qui peuvent utiliser les services WCF. Dans les applications Web de style AJAX, le client (par exemple, le navigateur d'une application Web) échange des petites quantités de données avec le serveur à l'aide de demandes asynchrones. L'intégration avec les fonctionnalités AJAX dans ASP.NET permet de générer facilement des services Web WCF accessibles à l'aide du JavaScript client du navigateur. Pour plus d'informations, consultez Intégration d'AJAX et prise en charge de JSON. Pour obtenir des exemples de code, consultez Exemples AJAX. Modèle de développement ASP.NET MVC 1.0.

Windows Forms : Améliorations de la fonctionnalité ClickOnce (déploiment).

Windows Presentation Foundation (WPF) : Dans le .NET Framework 3.5, Windows Presentation Foundation contient des modifications et des améliorations dans de nombreux domaines, y compris le

versioning, le modèle d'application, la liaison de données, les contrôles, les documents, les annotations et les éléments d'interface utilisateur 3D.

Windows Workflow Foundation : Intégration WF et WCF — Workflow Services Le .NET Framework 3.5 unifie les infrastructures Windows Workflow Foundation (WF) et Windows Communication Foundation (WCF) afin que vous puissiez utiliser WF pour créer des services WCF ou exposer votre flux de travail WF existant en tant que service. Cela vous permet de créer des services pouvant être persistants, pouvant transférer facilement des données à l'intérieur et à l'extérieur d'un flux de travail et pouvant appliquer des protocoles au niveau de l'application. Pour plus d'informations, consultez Création de services de workflow et de services fiables. Pour obtenir des exemples de code, consultez Exemples de services de workflow (WF).

Le .NET Compact Framework version 3.5 étend la prise en charge des applications mobiles distribuées en incluant la technologie Windows Communication Foundation (WCF). Il ajoute également de nouvelles fonctionnalités de langage selon les commentaires de la communauté, par exemple de nouvelles API et LINQ, et améliore le débogage avec des outils et des fonctionnalités de diagnostic mis à jour.

1.2. Visual Studio 2010 SP1 et .NET 4.0

Depuis le 12 avril 2010, est disponible la version WPF de Visual Studio.

Le mot clé est : dynamique, qui se dérive par exemple dans le langage C# 4.0 et un DLR (dynamic language runtime).(

Amélioration de la navigation et de l'éditeur de code :

- Gestion du multi-écrans, le zoom du code,
- Amélioration de l'Intellisense,
- Highlighting References,
- Navigate To,
- Call Hierarchy,
- Generate From Usage,
- Intellisense Suggestion Mode,
- Le mode de débogage IntelliTrace,
- Nouveau système d'extension de Visual Studio : Extension Manager.

En Développement Web, améliorations du développement web avec les dernières versions des différentes plateformes : ASP.NET Webforms version 4 et ASP.NET MVC Version 2 (puis 3 aujourd'hui, 4 en béta).

Développement applications riches :

- Améliorations du développement d'applications riches
- avec WPF 4.0,

- Améliorations du designer,
- Support de Windows 7,
- De nouveaux contrôles,
- Améliorations du rendu graphique, ...
- En développement Silverlight, un designer Silverlight et l'intégration de Silverlight 3 (puis 4.0 aujourd'hui). Entity Framework 2.0 pour ADO.NET 4.0.

Nous l'avons dit, la ligne directrice pour C# 4 est une programmation dynamique. Les objets sont "dynamiques" dans le sens où leur structure et leur comportement ne sont pas cloisonnés par un type statique et que le compilateur sais comment compiler le programme. Quelques exemples :

- Les objets d'un langage de programmation dynamique tel que Python ou Ruby.
- Les objets COM auxquels on accède depuis IDispatch.
- Les types .NET auxquels on accède à travers la réflexion.
- Les objets avec une structure changeante, comme les objets DOM HTML.

C# reste un langage fortement typé statique, cependant les équipes de développement souhaitent augmenter l'interaction avec les objets et faire évoluer ensemble C# et Visual Basic. Elles veulent maintenir les caractéristiques de chaque langage, mais avoir toutes les nouveautés introduites dans les 2 langages en même temps. Si les langages sont différents dans le style, ils ne le sont pas dans les fonctionnalités.

Les nouvelles fonctionnalités en C# 4 peuvent être découpées en 4 groupes.

- Dynamic lookup
- Paramètres nommés et optionnels
- COM
- Variance

Nous ne présenterons que les deux premières dans ce document.

1.3. Visual Studio 2012

- Développement d'applications dans Visual Studio
- Nouveautés d'Application Lifecycle Management avec Visual Studio et Team Foundation Server
- Mise en route de Blend pour Visual Studio
- Blend pour Visual Studio (Centre de développement – Applications Windows Store)
- Conception pour Windows Phone (Expression Blend 4)
- .NET Framework 4.5 (Windows 8, multi-tâches, support HTML 5)

1.4. Visual Studio 2013

- Gestion de projets en mode agile. Ajout de CodeLens et intégration de GIT via TFS

1.5. Visual Studio 2015

- Applications mobiles multiplateformes en C# avec Xamarin pour Visual Studio
- Applications mobiles multiplateformes en HTML/JavaScript avec Apache Cordova
- ASP.NET 5 est une mise à jour majeure de MVC, WebAPI et SignalR, et s'exécute sur Windows, Mac et Linux.
- Microsoft .NET Framework 4.6 offre environ 150 nouvelles API et 50 API.
- C# 6

1.6. Visual Studio 2017

- Nouvelle page de démarrage de l'environnement de développement.
- IntelliSence plus intelligente.
- Analyse du code fournit en temps réel des notifications quand vous tapez du code dans l'éditeur de code.
- Icône « Exécuter jusqu'au clic » directement dans le code.

1.7. Visual Studio 2019

- Nouvelle page de démarrage de l'environnement de développement.
- Les développeurs C++ bénéficieront désormais d'une nouvelle version du compilateur MSVC et de nombreuses autres bibliothèques ont été ajoutées pour améliorer leur productivité. L'intégration de cMake va leur permettre à l'avenir de détecter et d'activer automatiquement les chaînes d'outils Vcpkg pour les projets cMake et aussi ils pourront également bénéficier des avantages de nouvelles fonctionnalités de IntelliSense. Des fonctionnalités ont été également ajoutées pour les développeurs C# 8.0, Python sans oublier les développeurs .NET.
- Les développeurs C# peuvent utiliser à présent les modèles récursifs et il est désormais possible de créer des sessions Visual Studio Live Share pour la collaboration sur du code Python. Toujours pour les développeurs Python, vous avez désormais le choix entre les différents interpréteurs Python grâce à une nouvelle barre d'outils pour le langage. Vous pouvez consulter la FAQ ou la note de version pour en savoir plus sur toutes les fonctionnalités de ce bêta test.
- Visual Studio Preview 2 apporte également des nouveautés pour les développeurs Web. L'article parle de la prise en charge du débogage JavaScript pour les tests unitaires ainsi que le débogage des applications ASP.NET dans le cas où vous utilisez des conteneurs dans vos développements.

L’équipe souligne aussi que cette version prend en charge les dernières images ASP.NET et .NET Core. D’autres outils comme le dernier compilateur dex Android (d8) pour les développeurs Android et Kubernetes sont également intégrés.

- « Les outils Visual Studio Kubernetes sont désormais intégrés à la charge de travail de développement Azure pour une installation facile. Cela ajoutera l'application de conteneur pour le modèle de projet Kubernetes à Visual Studio. Cela vous permet également d'ajouter la prise en charge de Kubernetes à une application ASP.NET Core existante. Après avoir ajouté le support Kubernetes, vous pouvez créer, exécuter et déboguer votre application dans un cluster Live Azure Kubernetes Service (AKS) avec Azure Dev Spaces », a expliqué l’équipe de Visual Studio.

2. Simplifications de base

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Simplifications de base	60 mn	Cours, travaux dirigés	Individuel ou par binome
Apprendre les types implicites et les propriétés auto-implémentées.			

2.1. Variable à type implicite

Les variables locales à une méthode peuvent avoir un type implicite.

Une telle variable est déclarée avec le mot clef "var".

Le compilateur infère le type effectif de la variable dès sa déclaration.

Dans tout le reste du corps de la méthode la variable est alors fortement typée par le type inféré.

Le type inféré peut être quelconque y compris un type anonyme.

```
private void Methode()
{
  var st = "type implicite";
  Console.WriteLine(s.GetType()); // System.String

  var x  = 1268 ;
  x = "bonjour" ;

  var u  = "abcde" ;

  var t  = new [] { -6, 15, 25, 8 };

  var y  = u.ToCharArray()
}
```

Les variables à type implicite peuvent être déclarées dans une boucle for ou une boucle foreach (...) sur une collection de données :

```
for (var k = 0; k < 100; k++)
{
       var i = k +10;
       int j = k - 10;
}
```

Les variables à type implicite peuvent être déclarées dans une instruction `using`:

```
using (var flux = new StreamWriter("c:\\temp\\fichier.txt"))
{
    flux.AutoFlush = true;
}
```

2.2. Propriétés auto-implémentées

Le compilateur peut créer automatiquement un champ privé support de la propriété.

```
Class ConsoleNET35
{
   // Ancienne formulation des propriétés (avec membre privé)
   private string version;

   public string Version
   {
       get { return version; }
       set { version = value; }
   }

   // Nouvelle formulation
   public static string Nom
   {
       get;
       set;
   }
}
```

Le mot clé `private` peut être rajouté pour les propriétés en lecture seule : `private set;` ou en écriture seule : `private get;`

3. Les initialiseurs

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Les initialiseurs	60 mn	Cours, travaux dirigés	Individuel ou par binome
Déclarer et initialiser facilement des tableaux, objets, ...			

3.1. Initialiseurs d’objets

Un initialiseur d’objet permet d’initialiser des champs ou des propriétés d’un objet lors de sa création sans faire appel explicitement au constructeur de la classe.

```
class Client
{
   public string Nom {get; set;}
   public string Prenom {get; set;}
   public int ClientID {get; private set;}
   public Client()
   {
      ClientID = 9999;
   }
}
```

`Client personne = new Client{Nom = "martin", Prenom ="luc"}` Ou aussi : `Client personnne = new Client(){ Nom = "martin", Prenom = "luc"}`

Fait implicitement appel au constructeur sans paramètre : Client()

Note

Si la classe Client ne contient pas une déclaration soit explicite, soit implicite du constructeur par défaut sans paramètre, le compilateur signale alors une erreur.

3.2. Initialiseurs de collections

Un initialiseur de collection évite lors de la création d’une collection, d’utiliser plusieurs fois la méthode Add pour ajouter des objets à la collection.

```
List <int> listeNbr = new List <int> { 12, 65, 98, 14, -8, 36 };
```

```
List<Type> listeTypes = new List<Type>()
{
    new Type() {Code = -1, Titre = "Tâche"},
    new Type() {Code = 1, Titre = "Nouvelle fonctionalité"},
    new Type() {Code = 2, Titre = "Amélioration"},
    new Type() {Code = 3, Titre = "Correction"},
};
```

Il est possible d'initialiser une collection, si le type de la collection le permet, avec `null`.

4. Les anonymes

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Les anonymes	45 mn	Cours, travaux dirigés	Individuel ou par binome
Connaître les types et méthodes anonymes			

4.1. Types anonymes

Un type anonyme est une classe comportant uniquement des propriétés publiques en lecture seulement (pas de méthodes, ni de champs, ni d'événements).

Le type réel n'est pas disponible au niveau du code source.

Le type de chacune des propriétés est automatiquement inféré par le compilateur.

Un objet de type anonyme est instancié par un initialiseur d'objet anonyme (initialiseur sans le nom du type).

Type anonyme engendré par un initialiseur d'objet anonyme : `new { Nom = "martin" , Prenom = "luc" , age = 35 }`

Dans cette instanciation, le compilateur a engendré un objet de type anonyme interne et un type .NET pour chacune des propriétés.

Les variables à type implicite peuvent être affectée par des initialiseurs de type anonyme.

```
var Etudiant = new { Nom = "LE BOSSONEC", Prenom = "Morvan Tugdual", Age = 35 };
Console.WriteLine(Etudiant.Nom + " " + Etudiant.Prenom);// LE BOSSONEC Morvan Tugd
Console.WriteLine(Etudiant.ToString()); // { Nom = LE BOSSONEC, Prenom = Morvan Tu
Console.WriteLine(Etudiant.GetType());  // &lt;&gt;f__AnonymousType0'3[System.Stri
// Etudiant.Age = 53;  => interdit, lecture seule après instanciation.
```

5. Les expressions lambda

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Les expressions lambda	10 mn	Cours, travaux dirigés	Individuel ou par binome
Découvrir une simplification des opérations de calculs ou de recherches			

Le lambda-calcul est un formalisme de représentation des fonctions.

Le l-calcul a exactement le même pouvoir d'expression que les machines de Turing.

Soit f un programme dépendant ou non de la variable x, on note lx.f la fonction qui à une variable d'entrée x associe f : `lx : x ---> f`

La notation lxyzu.f représente la fonction ayant pour paramètres d'entrées les variables x, y, z, u.

- En C# la lambda-expression x => f représente l'expression lx.f.
- Une lambda-expression comporte une partie paramètres d'entrée et une partie instructions : liste paramètres="" => { Bloc instructions="" }
- Le lambda opérateur => est utilisé pour séparer la partie paramètre de la partie instructions.

Une lambda-expression est une fonction anonyme. Exemple pour une lambda-expression à 3 paramètres, qui renvoie un entier :

```
( int k, string s, char carLu) =>
{
      if( s[0] == carLu)
          return k+1;
      else
          return k*10;
}
```

Autres exemples simples sur une liste d'entiers :

```
var tInt = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
var listeInt = tInt.Where(i => i >= 5); // Liste des entiers supérieures à 5
decimal somme = tInt.Where(i => i >= 5).Sum(); // Somme des entiers supérieures à
var compteur = tInt.Count(i => i > 5); // Nombre d'entiers supérieures à 5
```

Une lambda-expression produisant le même résultat que la précédente : `( int k, string s, char carLu) => s[0]==carLu ? k+1 : k*10`

Une l-expression peut être assignée à un delegate :

```
delegate int monDelegue ( int k, string s, char carLu)

myDel monDelegue = ( int k, string s, char carLu) =>
{
  if( s[0] == carLu)
      return k+1;
  else
      return k*10;
}

int k = unDelegue ( 8, "abcde" , 'x' );
```

6. Les extensions

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Les extensions	10 mn	Cours, travaux dirigés	Individuel ou par binome
Comment faire pour rajouter des méthodes à des classes existantes non accessibles			

Permet d'étendre un type sans recompiler le type

Une méthode d'extension d'un type, est une méthode statique spéciale, qui est appelée dans le code client comme si elle faisait partie des méthodes initiales du type.

Une méthode d'extension d'un type est appelée comme si elle était une méthode d'instance, c'est le compilateur qui se charge de la traduction en MSIL.

Une méthode d'extension permet d'étendre une classe ou une interface, mais de la redéfinir.

Le mot clef `this` devant le premier paramètre indique la classe que l'on veut étendre (ici nous rajoutons ainsi deux méthodes d'extensions à la classe string : `nbrMots()` et `nbrVoyelles()` :

```
public static class MasterChainesExtension
{
    public static int NombreMots( this string ch)
    {
        return ch.Split(new char[] { ' ', ',', ';', '.', '!', '?' },
        StringSplitOptions.RemoveEmptyEntries).Length;
    }
```

```
    public static int NombreVoyelles( this string ch)
    {
        int nbrVoy=0;
        for ( int i =0; i<ch.Length; i++ )
            if ( ch[i]=='a' |ch[i]=='e' |ch[i]=='i'
            |ch[i]=='o' |ch[i]=='u' |ch[i]=='y'  )
                nbrVoy++;
        return nbrVoy;
    }
}
```

Les deux méthodes `NombreMots` et `NombreVoyelles` sont maintenant ajoutées à la liste des méthodes de la classe `sealed string`. Les méthodes d'extensions s'utilisent comme les autres méthodes d'instances.

7. Appel dynamique des méthodes

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Appel dynamique des méthodes	10 mn	Cours, travaux dirigés	Individuel ou par binome
Comment appeler des méthodes non disponibles lors de la compilation ?			

7.1. Recherche dynamique

http://msdn.microsoft.com/en-us/library/dd264736(VS.100).aspx Dans les langages interprétés, ou dynamiques, du fait des types qui sont indéterminées (à typage dynamique), il n'y a pas de phase de génération d'un exécutable, préalable à l'exécution, ce qui peut leur conférer une meilleure réactivité. La contre partie est qu'il n'y a pas de détection syntaxique ou de vérification de type, elle est effectuée pendant l'exécution.

Un mot clé `dynamic` est introduit dans C#4, qui permet d'écrire des méthodes, des opérateurs et des appels d'indexeur, des propriétés et des champs internes, dont l'invocation sera résolue à l'exécution.

Ce n'est donc plus le CLR, mais le DLR (Dynamic Language Runtime). En utilisant le mot clef dynamic en C#4, le compilateur va ajouter des appels à la librairie `Microsoft.CSharp.dll`, qui est une librairie qui utilise le DLR pour faire communiquer le monde statique de C# avec celui dynamique.

Cela se traduit par cet exemple :

Le code suivant compile en C# 4.0 alors que RechercheParNom() n'existe pas :

```
dynamic classe = new MaClasse();
classe.RechercheParNom();
```

Le code suivant ne compilera pas en C# 3.5, la méthode appelée sera introuvable pour le compilateur :

```
var classe = new MaClasse();
classe.RechercheParNom();
```

```
dynamic  d = 7 ; // conversion implicite
var testSomme = d + 3;
System.Console.WriteLine(testSomme); // testSomme est également dynamic !
```

7.2. Opérations dynamiques

Comme dit plus haut, ce n'est pas seulement l'appel des méthodes qui peut être invoqué dynamiquement. Les champs et les propriétés, les indexeurs et les opérateurs et même l'invocation des délégués peuvent être invoqués dynamiquement.

```
Dynamic d = GetDynamicObject(…) ;
d.M(7) ; // Appel de méthode
d.f = d.P ; // Getter et Setter , champs et propriétés
d[‘one’ ] = d[‘two’] ; // Getter et Setter d’indexeurs
int i = d + 3; // Appel d’opérateurs
string s = d(5,7) ; // invocation de delegate.
```

Ici, le rôle du compilateur C# 4.0 est simplement de fournir les informations nécessaires : "Que peut faire d". Ensuite le runtime peut déterminer exactement le comportement de l’objet d. Le résultat d’une opération dynamique sera du type "dynamic"

Aller au guide de programmation sur MSDN. [http://msdn.microsoft.com/en-us/library/dd264736(VS.100).aspx]

8. Paramètre nommé et optionnel

Déroulement

Phase	**Durée**	**Type de travail**	**Modalités de travail**
Paramètre nommé et optionnel	10 mn	Cours, travaux dirigés	Individuel ou par binome
Comment pouvoir passer des paramètres dans le désordre ou partiellement ?			

Les paramètres nommés et optionnels sont deux fonctionnalités distinctes, mais ils sont souvent utilisés ensemble. Le paramètre optionnel permet d'omettre un argument à l'invocation d'une méthode. Beaucoup D'API COM comme Office Automation sont écrites avec des paramètres nommés et optionnels. Il peut être très contraignant d'appeler des API en C#, car il faut poser explicitement les arguments et ceux-ci sont souvent des valeurs par défaut et peuvent donc être omis. Paramètres optionnels : Un paramètre est déclaré optionnel simplement en lui affectant une valeur par défaut.

```
public void ControleIdentifiantPourAction(int identifiant, int action = 5,
int controle = 7){...}
```

Ici `action` et `controle` sont des paramètres optionnels et peuvent être omis lors de l'appel.

```
ControleIdentifiantPourAction(1,2,3) ; // Appel de ControleIdentifiantPourAction
ControleIdentifiantPourAction(1,2) ;   // Appel de ControleIdentifiantPourAction(
ControleIdentifiantPourAction(1) ;     // Appel de ControleIdentifiantPourAction(
```

C# 4.0 ne permet pas d'omettre un paramètre comme dans ce cas : `ControleIdentifiantPourAction(1,,3).` Ceci rendrait le code illisible. A la place, on passera par les arguments nommés. Si vous voulez omettre seulement le paramètre y en appelant `ControleIdentifiantPourAction` on pourra écrire :

```
ControleIdentifiantPourAction(1, controle : 3) ; // en passant controle par son n
ControleIdentifiantPourAction(identifiant : 1, controle :3) ;// en passant identi
ControleIdentifiantPourAction(controle : 3, identifiant : 1) ;// cette écriture a
```

Les paramètres nommés et optionnels peuvent être utilisés non seulement avec des méthodes mais aussi avec les indexeurs et les constructeurs.

9. Nouveautés de C# avec .NET 4.5

Mise en place de mots clés pour appels parallèles

```
public async Task<int> MethodAsync(string input)
{
    return -1;
}

// Appel
public async Task Main(string input)
{
    return await MethodAsync(input);
}
```

```
// Boucle parallèles
Parallel.ForEach(files, currentFile =>
                                {
        // action
        });
```

10. Nouveautés du C# 6

Initialiseurs de propriétés automatiques et propriétés automatiques en lecture seule

```
public int Age { get; set; } = 42;
public int Age { get; } = 42; // Cas de la lecture seule.
```

Imports statiques, pour éviter d'ajouter l'espace de nom dans les classes statiques

```
using static System.Math;
```

Simplification des propriétés retournant une expression

```
public double Distance
{
    get
    {
        return Sqrt(X * X + Y * Y);
    }
}
```

devient
:

```
public double Distance => Sqrt(X * X + Y * Y);

pu
```

blic Point Move(int dx, int dy) => new Point(X + dx, Y + dy);

Initialiseurs de dictionnaires et de membres indexés

```
var letters = new Dictionary<string, int>
{
    { "a", 1 },
    { "b", 2 },
    { "c", 3 },
    ...
};
```

devient

```
var letters = new Dictionary<string, int>
{
    ["a"] = 1,
    ["b"] = 2,
    ["c"] = 3,
    ...
};
```

await dans les blocs catch et finally

Filtres d'exception

ex :

```
catch(IOException ex) if (ex.HResult == 0x80070020)
{
    // File is in use by another process
    // Handle that case...
}
```

Propagation de null

Nouvel opérateur :

```
string name = customer != null ? customer.Name : null;
```

devient
:

```
string name = customer?.Name;
```

Nouveau formatage des chaines

```
string s = string.Format("Bonjour {0}, nous sommes le {1:D}", user.Name, DateTime.
```

devient

```
string s = $"Bonjour {user.Name}, nous sommes le {DateTime.Today:D}";
```

Opérateur nameof

```
string s = nameof(Console.Write); // renvoie "Write"
```

11. Nouveautés du C# 7

Fonctions locales

C# 7 introduit donc une fonctionnalité qui existe déjà dans la plupart des langages fonctionnels : les fonctions locales. Il s'agit tout simplement de la possibilité de déclarer une méthode à l'intérieur d'une autre méthode. Ces fonctions locales ne sont accessibles que dans le scope de la méthode qui les déclare. Par exemple :

```
 Monster GetClosestTarget(Point playerPosition, Monster[] monsters)
{
    double GetDistance(Point a, Point b)
    {
        double dx = a.X - b.X;
        double dy = a.Y - b.Y;
        return Math.Sqrt(dx * dx + dy * dy);
    }
```

```
    return monsters
        .OrderBy(m => GetDistance(playerPosition, m.Position))
        .FirstOrDefault();
}
```

Le fait de grouper les fonctions auxiliaires avec la méthode qui les utilise n'est pas le seul intérêt de cette approche ; en effet, les fonctions locales peuvent accéder directement aux variables locales et paramètres de la fonction qui les déclare. On pourrait donc par exemple simplifier le code ci-dessus comme suit :

```
Monster GetClosestTarget(Point playerPosition, Monster[] monsters)
{
    double GetDistance(Point p)
    {
        double dx = playerPosition.X - p.X;
        double dy = playerPosition.Y - p.Y;
        return Math.Sqrt(dx * dx + dy * dy);
    }

    return monsters
        .OrderBy(m => GetDistance(m.Position))
        .FirstOrDefault();
}
```

Tuples

Retour de méthode en introduisant les tuples. Un tuple est un ensemble ordonné fini de valeurs typées (potentiellement de types différents), et éventuellement nommées. Dit comme ça, ce n'est pas très parlant, on va donc voir un exemple. La fonction suivante calcule et renvoie le nombre d'éléments dans une liste de nombres, ainsi que leur somme :

```
  static (int count, double sum) Tally(IEnumerable<double> values)
{
    int count = 0;
    double sum = 0.0;
    foreach (var value in values)
    {
        count++;
        sum += value;
    }
    return (count, sum);
}


var (count, sum) = Tally(numbers);
```

```
Console.WriteLine($"Il y a {count} nombres dans la liste, et leur somme est {sum}.
```

Déconstructeurs

Le mécanisme de décomposition mentionné plus haut pour les tuples n'est en fait pas limité aux tuples : n'importe quel type peut être décomposé de cette manière, s'il a une méthode Deconstruct avec la signature adéquate. Par exemple, pour décomposer un type Point en ses propriétés X et Y, on peut lui ajouter une méthode comme celle-ci :

```
  public void Deconstruct(out double x, out double y)
{
    x = this.X;
    y = this.Y;
}
 // Utilisation
Point p = GetPosition();
var (x, y) = p;
```

Pattern matching

Le pattern matching (qu'on peut traduire par « filtrage par motif » selon Wikipedia) est un mécanisme présent dans la plupart des langages fonctionnels, et qui permet de déterminer si une valeur correspond à certains cas prédéfinis. Cela ressemble à un switch, et effectivement c'est un peu similaire, mais beaucoup plus puissant, puisqu'il permet de faire ce genre de choses (exemple en F#) :

```
if (shape is Rectangle r)
    Console.WriteLine($"Rectangle with area {r.Width * r.Height}");
else if (shape is Circle c && c.Radius < 5)
    Console.WriteLine("Small circle");
else if (shape is Circle c)
    Console.WriteLine("Large circle");
```

```
switch (shape)
{
    case Rectangle r:
        Console.WriteLine($"Rectangle with area {r.Width * r.Height}");
        break;
    case Circle c when c.Radius < 5:
        Console.WriteLine($"Small circle");
        break;
    case Circle c:
        Console.WriteLine($"Large circle");
        break;
}
```

Une conséquence de l'ajout du pattern matching à C# est que l'instruction switch a été améliorée. Jusqu'ici, elle était limitée aux types primitifs et enums, mais on peut maintenant l'utiliser sur des valeurs de n'importe quel type.

Variables out

C# 7 permettra de déclarer la variable directement dans l'appel, de la façon suivante :

```
if (int.TryParse(s, out int i))
{
    Console.WriteLine($"La chaine représente un entier de valeur {i}");
}
```

Il est également possible d'ignorer la valeur du paramètre out, si on n'en a pas besoin. Pour cela, il suffit d'utiliser la syntaxe out _ :

```
  if (int.TryParse(s, out _))
{
    Console.WriteLine("La chaine représente un entier");
}
```

Amélioration des littéraux numériques

C# 7 apporte deux améliorations à l'écriture de littéraux numériques. La notation binaire. Il était jusqu'ici possible d'écrire des nombres entiers en décimal ou hexadécimal, on peut maintenant les écrire aussi en binaire, ce qui est pratique pour certains cas d'usage (manipulation de bits par exemple). Il suffit pour cela de préfixer le nombre par 0b :

```
byte binary = 0b0001_1000;
int hex = 0xAB_BA;
int dec = 1_234_567_890;
```

Membres dont le corps est une expression

C# 6 avait introduit une syntaxe simplifiée pour les méthodes et propriétés constituées d'une seule instruction :

```
public double Length => Math.Sqrt(X*X + Y*Y);
public string ToString() => $"({X}, {Y})";
```

Mais cette syntaxe ne pouvait pas être appliquée aux constructeurs, finaliseurs et aux accesseurs de propriétés. C# 7 généralise cette syntaxe, on peut donc maintenant écrire des choses comme ça :

```
  public Person(string name) => _name = name;

~Person() => Console.WriteLine("Finalized");

public string Name
{
    get => _name;
    set => _name = value;
}
```

Expressions throw

En C# 7, throw devient une expression convertible en n'importe quel type. Cela permet par exemple de simplifier la validation des arguments dans un constructeur :

```
  public Person(string firstName, string lastName)
{
    FirstName = firstName ?? throw new ArgumentNullException(nameof(firstName));
    LastName = lastName ?? throw new ArgumentNullException(nameof(lastName));
}
```

Généralisation du type de retour des méthodes asynchrones

C# 5 avait introduit le support du code asynchrone directement dans le langage. Une méthode marquée async ne pouvait avoir comme type de retour que void, Task ou Task(T). C# 7 permettra de définir des méthodes asynchrones renvoyant d'autres types, pour peu que ces types répondent à certaines caractéristiques.

```
static async ValueTask<decimal> GetPriceAsync(int productId)
{
    if (_cache.TryGetValue(productId, out decimal price))
        return price;

    price = await FetchPriceFromServerAsync(productId);
    _cache[productId] = price;
    return price;
}
```

Variables locales et retours de fonctions par référence

C# permet depuis toujours de passer des paramètres par référence plutôt que par valeur, grâce à l'utilisation du mot clé ref (et out, qui est en fait la même chose avec une sémantique un peu différente). C# 7 étend ce mécanisme aux variables locales et aux valeurs de retour des fonctions, évitant ainsi les copies inutiles de données dans certains scénarios.

```
   struct MyBigStruct
{
    public int Id { get; set; }
    public string Name { get; set; }
}

static MyBigStruct[] _items;
static ref MyBigStruct FindItem(int id)
{
    for (int i = 0; i < _items.Length; i++)
    {
        if (_items[i].Id == id)
        {
            return ref _items[id];
        }
    }
    throw new Exception("Item not found");
}

static void Test()
{
    ref MyBigStruct item = ref FindItem(42);
    item.Name = "test";
}
```

Dans la méthode Test, on récupère non pas une copie, mais une référence vers la structure, là où elle se trouve dans le tableau _items. Ainsi, quand on modifie la valeur de Name, c'est bien l'objet dans le tableau qui est modifié (c'est en fait le même comportement qu'on aurait eu si MyBigStruct avait été un type référence).

Cette nouvelle fonctionnalité peut d'ailleurs donner du code un peu surprenant ; en effet, le code suivant est parfaitement valide :

```
FindItem(42) = new MyBigStruct();
```

12. Nouveautés du C# 8

Les types références Nullables

On active les références nullables avec #nullable enable

Les flux asynchrones

Maintenant, il est possible de créer des méthodes qui retournent un flux asynchrone et le consommer.

La méthode qui émet le flux doit respecter trois conditions :

• être déclarée avec le modificateur « async »

• retourner un objet de type IAsyncEnumerable(T)

• contenir une instruction « yield return » pour pouvoir retourner des éléments consécutifs

Pour ensuite pouvoir consommer ce flux, il faut ajouter le mot-clé « await » devant une boucle « foreach » lorsqu'on énumère les différents éléments constitutifs du flux. Et puisqu'on on utilise le mot-clé « await », la méthode doit être déclarée avec le modificateur « async » et retourner un type d'objet autorisé comme Task ou Task(TResult).

```
class Program
    {

        public static async IAsyncEnumerable<int> download()
        {
            for (int i = 1; i <= 100; i++)
            {
                Random rand = new Random();
                await Task.Delay(rand.Next(1,10)*100);
                yield return i;
            }
        }

        static async Task Main(string[] args)
        {
            await foreach (var number in download())
            {
                Console.Clear();
                Console.Write("Donwloading.... : "+number+"%");
            }
        }
    }
```

Les Index et plages de données

• pour récupérer le premier élément on doit faire : array[0]

• pour récupérer le dernier élément on doit faire : array[^1]

Attention, si on utilise l'index « 0 » pour spécifier le premier élément d'un tableau, ça ne fonctionne pas de la même manière lorsqu'on part de la fin. En effet, «^0 » correspond à la fin du tableau, soit l'index qui suit le dernier élément. C'est surtout à utiliser lorsqu'on manipule des plages de données et non pas pour extraire une donnée.

Pour manipuler les plages, c'est tout aussi simple. Exemple avec un tableau de dix éléments :

• pour récupérer les trois premiers éléments d'un tableau : array[0..3]

• pour récupérer les trois derniers : array[8..11]

On peut aussi utiliser l'opérande ^ et faire les plages en partant de la fin :

• pour récupérer les trois premiers : array[^8..^11]

• pour récupérer les trois derniers : array[^3..^0]

Et si vous souhaitez récupérer l'intégralité des données : array[..]

Il est même possible de créer un objet Range et l'appliquer à un tableau :

Range troisPremiers = 0..3 ;

var test = array[troisPremiers] ;

```
class Program
    {
        static void Main(string[] args)
        {
            var jours = new string[]
            {
               "lundi",
               "mardi",
               "mercredi",
               "jeudi",
               "vendredi",
               "samedi",
               "dimanche"
            };

            // Index
            Console.WriteLine("Premier élément : " + jours[0]);
            Console.WriteLine("Deuxiemme élément : " + jours[1]);
            Console.WriteLine("Dernier élément : " + jours[^1]);
            Console.WriteLine("Deuxiemme élément en partant de la fin: " + jours[^
            Console.WriteLine();

            // Plages
            var first3 = jours[0..3]; // lundi, mardi, mercredi
            var last3 = jours[4..7]; // vendredi, samedi, dimanche

            var first3ByLast = jours[^7..^4]; //lundi, mardi, mercredi
            var last3ByLast = jours[^3..^0]; //vendredi, samedi, dimanche

            var tousLesJours = jours[..];

            Range mardiMercredi = 1..3;
            var marMer = jours[mardiMercredi]; // mardi, mercredi

            Console.WriteLine("Les trois premiers éléments sont : ");
            foreach (var item in first3)
            {
```

```
            Console.WriteLine(item);
        }
        Console.WriteLine();

        Console.WriteLine("Les trois derniers éléments sont : ");
        foreach (var item in last3)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine();

        Console.WriteLine("Tous les éléments sont : ");
        foreach (var item in tousLesJours)
        {
            Console.WriteLine(item);
        }
        Console.WriteLine();

    }
}
```

L’expression Switch

Permet maintenant d’utiliser une syntaxe plus concise et avec moins de répétition de code. Mais comme il est difficile de vous expliquer cette nouvelle approche avec des mots, alors partons plutôt sur un exemple avec l’énumération Plateformes qui liste des plateformes

```
public enum Plateformes
       {
           Playstation,
           Xbox,
           Switch,
           PC,
           Stadia
       }

public static string ReactionJoueur(Plateformes plateforme) =>
        plateforme switch
           {
               Plateformes.Playstation =>  "La meilleure des consoles !!!!!!",
               Plateformes.Xbox => "Trop aucun interet",
               Plateformes.Switch => "Pour les enfants",
               Plateformes.PC => "Carte graphique ",
               _ => "Inconnue",
       };
```

13. Nouveautés du C# 9

Enregistrements

Setter init uniquement

Instructions de niveau supérieur

Améliorations des critères spéciaux

Entiers dimensionnés natifs

Pointeurs fonction

Supprimer l'émission de l'indicateur localsinit

Nouvelles expressions typées cibles

fonctions anonymes statiques

Expressions conditionnelles typées cible

Types de retour covariant

GetEnumeratorPrise en charge des extensions pour les foreach boucles

Paramètres d'abandon lambda

Attributs sur des fonctions locales

Initialiseurs de module

Nouvelles fonctionnalités pour les méthodes partielles

Types d'enregistrements

C# 9,0 introduit *types d'enregistrements _, qui sont un type référence qui fournit des méthodes synthétisées pour fournir une sémantique de valeur pour l'égalité. Les enregistrements sont immuables par défaut. Les enregistrements prennent en charge l'héritage.

```
    public record Person
{
    public string LastName { get; }
    public string FirstName { get; }

    public Person(string first, string last) => (FirstName, LastName) = (first, la
}
```

Setter init uniquement

*Les Setters init uniquement fournissent une syntaxe cohérente pour initialiser les membres d'un objet. Les initialiseurs de propriété permettent de savoir clairement quelle valeur définit la propriété.

```
    public struct WeatherObservation
{
```

```
    public DateTime RecordedAt { get; init; }
    public decimal TemperatureInCelsius { get; init; }
    public decimal PressureInMillibars { get; init; }

    public override string ToString() =>
        $"At {RecordedAt:h:mm tt} on {RecordedAt:M/d/yyyy}: " +
        $"Temp = {TemperatureInCelsius}, with {PressureInMillibars} pressure";
}
```

Instructions de niveau supérieur

Les instructions de niveau supérieur suppriment la cérémonie inutile de nombreuses applications.

```
    System.Console.WriteLine("Hello World!");
```

Améliorations des critères spéciaux

```
    public static bool IsLetterOrSeparator(this char c) =>
    c is (>= 'a' and <= 'z') or (>= 'A' and <= 'Z') or '.' or ',';

   if (e is not null)
{
    // ...
}
```

Fonctionnalités d’ajustement et de fin

La plupart des autres fonctionnalités vous aident à écrire du code plus efficacement. En C# 9,0, vous pouvez omettre le type dans une new expression lorsque le type de l’objet créé est déjà connu. L’utilisation la plus courante est dans les déclarations de champ :

```
    private List<WeatherObservation> _observations = new();

    WeatherStation station = new() { Location = "Seattle, WA" };
```

14. LINQ

Déroulement

Phase	**Durée**	**Type de travail**	**Modalités de travail**
Linq	60 mn	Cours, travaux dirigés	Individuel ou par binome
Découvrir la puissance du langage de requète intégré de la plateforme .NET			

14.1. Language INtegrated Query (LINQ - C# 3.0)

LINQ est un sur-ensemble d'extension des langages C# et VB.Net sur la plateforme .Net 3.5.

Les extensions apportées par LINQ se situent au niveau de la liaison entre les données structurées et les objets.

LINQ apporte une ensemble d'opérateurs de requêtage sur des structures de données de types différents :

- Requêtes sur des collections .Net
- Requêtes sur des données XML
- Requêtes sur des DataSet Ado.Net
- Requêtes sur des BD SQL serveur

Les outils de C# 3.0 pour la technologie LINQ :

- Une syntaxe proche de celle du SQL.
- Les types anonymes.
- Le typage implicite de variable.
- Les lambda-expressions.
- Les expressions de requêtes.

14.2. LINQ expressions de requêtes

Les opérateurs LINQ

```
from, where, in, select, orderby, ascending, descending, group, by, into, take, takeWhile, skip
```

Une instruction de requête LINQ est une affectation dont la partie droite commence obligatoirement par un from ... in ... ; la partie gauche est une variable de type IEnumerable<T> qui contient le résultat de la requête :

```
< id.variable > = from < id.variable parcours > in < source données >
```

Note

Le résultat de la requête est toujours du type IEnumerable<T>.

La partie droite d'une instruction de requête LINQ se termine obligatoirement soit par l'opérateur select, soit par l'opérateur group.

```
< id.variable > = from < id.variable parcours>  in  < source données >
                          select < id.variable parcours> ;
```

```
< id.variable > = from < id.variable parcours>  in  < source données >
                  group < id.variable parcours> by  < clef >;
```

LINQ requêtes sur un tableau .Exemple d'opération LINQ sur un tableau d'int :

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };

IEnumerable<int> nombrePairs =
    from int nombre in table
    where nombre % 2 == 0
    select nombre ;

int[] tablePair = nombrePairs.ToArray( );
```

Même exemple que le précédent sur le tableau d'int (Variable à type implicite) :

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };

var nombresPairs =
    from nombre in table
    where nombre % 2 == 0
    select nombre ;

int[] tablePair = nombresPairs.ToArray( );
```

LINQ requêtes sur une List<T>

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
List<int> listeInt = new List<int>(table);

var nombresPairs =
    from nombre in listeInt
```

```
    where nombre % 2 == 0
    select nombre ;

List<int> listePair = nombresPairs.ToList( );
```

LINQ requêtes sur un ArrayList

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
ArrayList arraylisteInt = new ArrayList (table);

var nombresPairs =
    from int nombre in arraylisteInt
    where nombre % 2 == 0
    select nombre ;

ArrayList tablePair = new ArrayList ( nombresPairs.ToArray( ) );
```

14.3. LINQ des méthodes d’extension

Situées dans IEnumerable<T> et IEnumerable

Méthode d’extensions à exécution immédiate (aggrégateurs)

```
Sum() : public static decimal Sum( this IEnumerable<decimal> source )
Average() : public static decimal Average( this IEnumerable<decimal> source )
Last() : public static T Last<T>( this IEnumerable<T> source )
First() : public static T First<T>( this IEnumerable<T> source )
Max() : public static T Max<T>( this IEnumerable<T> source )
Min() : public static T Min<T>( this IEnumerable<T> source )
```

Exemple de méthodes d’extensions à exécution immédiate :

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
int n = table.Sum();
n = table.Last();
n = table.First();
n = table.Max();
n = table.Min();
double dMoyenne = table.Average();
```

Méthodes d'extensions à exécution immédiate avec lamda-expressions

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
int n0 = table.Sum(n => 3*n);
n0 = table.Sum ( n =>
    {
        if (n % 2 == 0)
             return 0;
        else
             return n;
   }
  );
double dMoyenne = table.Average(n => 2*n);
```

14.4. LINQ méthodes d'extension déferrées

Une méthode de requête LINQ est dite à exécution déferrée lorsqu'elle retourne un objet qui contient toutes les informations utiles à la requête et que cette requête n'est exécutée que lors de l'appel de l'énumateur de l'objet GetEnumerator et en particulier à travers une boucle foreach.

Méthodes d'extension à exécution déferrée : la projection Select, Le filtrage Where, L'union ensembliste Union, ...

Méthode d'extension à exécution déferrée : la projection Select

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };

IEnumerable<int> t = table.Select (n => n / 2);

foreach (var nb in t)
      Console.Write(nb + ", ");
```

Dans une instruction de requête LINQ l'opérateur select est traduit par le compilateur par un appel à la méthode déferrée Select().

Méthode d'extension à exécution déferrée : le filtrage Where

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };

IEnumerable<int> u = table.Where (n => n % 2 == 0);
```

```
foreach (var nb in u)
     Console.Write(nb + ".");
```

Dans une instruction de requête LINQ l'opérateur where est traduit par le compilateur par un appel à la méthode déferrée Where(), l'appel à l'énumérateur est effectué par l'instruction foreach.

Méthode d'extension à exécution déferrée : l'union ensembliste Union

```
int[] table1 = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
int[] table2 = new int[10] { -3, 9, 58, 64, -99 };

IEnumerable<int> u = table1.Union ( table2 );

foreach (var nb in u)
     Console.Write(nb + ".");
```

Méthode d'extension à exécution déferrée : projection tantque TakeWhile

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };

IEnumerable<int> u = table1.TakeWhile ( n => n < 60 );

foreach (var nb in u)
     Console.Write(nb + ".");
```

Les expressions de requêtes sont à exécution déferrée

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
ArrayList arraylisteInt = new ArrayList (table);

var nbrPairs =
    from int nbr in arraylisteInt
    where nbr % 2 == 0
```

```
    select nbr ;

ArrayList tablePair = new ArrayList (nbrPairs.ToArray());
```

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
List<int> listeInt = new List<int>(table);

var nbrPairs =
    from nbr in listeInt
    where nbr % 2 == 0
    select nbr ;

List<int> listePair = nbrPairs.ToList( );
```

ToList() lance l'exécution immédiate de la requête

L'exécution des expressions de requêtes peuvent être forcées par ToList() ou ToArray() :

```
int[] table = new int[10] { -5, 9, 58, 64, 100, 57, 14, 36, 11, 8 };
List<int> listeInt = new List<int>(table);

var nbrPairs =
    (from nbr in listeInt
    where nbr % 2 == 0
    select nbr).ToList() ;

List<int> nbrPairs =
    (from nbr in listeInt
    where nbr % 2 == 0
    select nbr).ToList();
```

15. Travaux dirigés

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Travaux dirigés	60 mn	Cours, travaux dirigés	Individuel ou par binome

Phase	Durée	Type de travail	Modalités de travail
Mise en pratique des techniques apprises			

15.1. Réalisation d'une classe d'extension pour une méthode de comptage et d'extraction d'une chaine

```
/// <summary>
/// Classe d'Extension de la classe string.
/// </summary>
public static class ExtensionChaine
{
    public static string ExtraitCompte(this string chaine)
    {
        if ( string.IsNullOrEmpty(chaine) )
            return "0";

        var longueur = chaine.Length;
        if ( longueur > 3 )
            return chaine.Substring(0, 3) + longueur.ToString();
        else
            return chaine + longueur.ToString();
    }
}
```

Usage : `string s = "master cci".ExtraitCompte(); // s = mas10`

15.2. Exercices sur Linq appliqués à LinqPad et Master.HelpDesk.

Exemple réalisé (hors Contacts) :

```
 var tInt = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

// Requete sur la liste d'entiers
 var listeIntLinq = from i in tInt
                    where (i >= 5)
                    orderby i descending
                    select new { Double = i * 2, Triple = i * 3 };

 // Requete linq formulée avec expressions Lambda
 var listeIntLambda = tInt
                      .Where(i => i >= 5)
                      .OrderByDescending(i => i)
```

```
.Select(i => new { Double = i * 2, Triple = i * 3 }); ;
```

```
foreach ( var item in listeIntLinq )
{
    Console.WriteLine(item.Double + " " + item.Triple); // 18 27,  16 24, ...
}
```

15.3. La configuration d'une application .NET (Settings, ConfigurationManager)appliquée à Master.HelpDesk.

Il est conseillé de réaliser la configuration d'une application à partir de paramètres stockés hors code, afin de ne pas avoir à recompiler à chaque changement d'une chaine de connexion à une base de données, de couleurs d'écran, de combinaisons de lancement, etc... Ici nous voyons l'utilisation d'un fichier de configuration XML (.config) qui a l'avantage d'être manipulable et facilement accessible.

Au préalable, rajouter un nouvel élément fichier de configuration d'application (`App.config`). Lors de la compilation, il sera copié sous le nom `NomApplication.exe.config` dans le répertoire `bin\debug`. Puis ajouter la référence à `System.Configuration`.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
</configuration>
```

Il existe deux sources de paramétrages :

- Dans les propriétés de votre projet, allez dans Settings, et ajouter des valeurs, application ou utilisateur : Exemple : `Identifiant String admin` pour utilisateur et `AutoRun bool adTrue` pour l'application (important : les propriétés Application ne sont pas modifiables directement à l'exécution).

  L'utilisation de ces valeurs s'effectuera simplement dans votre code par :

```
if ( Properties.Settings.Default.AutoRun ){...}

Properties.Settings.Default.DerniereExecution = DateTime.Now;
Properties.Settings.Default.Save(); // Mise à jour du ficher .config
```

- L'autre méthode est de rajouter les paramètres dans les balises du fichier app.Config ainsi :

```
<connectionStrings>
  <add name="ConnexionSqlite" providerName="system.data.sqlite" connectionString
</connectionStrings>
```

```
<appSettings>
  <add key="CheminApplication" value="c:\program files\Contacts\" />
</appSettings>
```

L'accès en exécution se fait par exemple ainsi :

```
// Lecture autres paramètres
ConfigurationManager.AppSettings["CheminApplication"] = Directory.GetCurrentDire
ConnectionStringSettings chaineConnexionSqlite = ConfigurationManager.Connection

// Ecriture
Configuration configuration = ConfigurationManager.OpenExeConfiguration(Configur
configuration.AppSettings.Settings.Remove("AncienParametre");
configuration.AppSettings.Settings.Add("NouveauParametre", "Valeur du nouveau pa
configuration.Save(ConfigurationSaveMode.Modified);
ConfigurationManager.RefreshSection("appSettings");
```

15.4. Créer l'installateur d'un programme .NET, le configurer et le déployer.

Visual Studio permet de créer des fichiers MSI (Microsoft Installer) pour l'installation des applications. Chaque fichier MSI est une sorte de base de données et contient des informations de configuration et des fichiers compressés. Les fichiers MSI sont traités avec l'application `msiexec.exe`.

Exemple : Pour installer une application à partir de `C:\Master.HelpDesk.msi`, vous pouvez double cliquer sur ce fichier ou alors en ligne de commandes, lancer le journal d'installation en tapant :

```
msiexec /l* Contacts.log /i C:\Master.HelpDesk.msi
```

Réalisation du projet d'installation :

1. Démarrez un nouveau projet en appliquant l'une des méthodes suivantes : Dans le menu Fichier, pointez sur Nouveau, puis cliquez sur Projet. ou Si vous disposez d'un projet ouvert pour lequel vous souhaitez créer un package de configuration, cliquez avec le bouton droit sur Solution Mon-Projet (où le nom de votre projet est MonProjet) dans l'Explorateur de solutions, pointez sur Ajouter, puis cliquez sur Nouveau projet. Dans la boîte de dialogue Nouveau projet, sélectionnez Projets de configuration et de déploiement dans le volet Type de projet, puis sélectionnez le type de configuration souhaité dans le volet Modèles.

   Le projet est ajouté à l'Explorateur de solutions et l'Éditeur du système de fichiers s'ouvre.

2. Dans la boîte de dialogue Propriétés, sélectionnez la propriété ProductName, puis tapez le nom de l'application et renseignez les champs de façon exhaustive car on les retrouve par exemple dans le panneau de configuration dans Ajout/Suppression de programmes.

3. Dans l'Éditeur du système de fichiers, sélectionnez le nœud Dossier d'application. Cliquez avec le bouton droit sur le dossier Application, puis dans le menu Action, cliquez sur Ajouter, puis sur

Fichier. Dans la boîte de dialogue Ajouter des fichiers, sélectionnez tous les fichiers que vous souhaitez ajouter à votre application (de préférence issus du dossier Release de l'application.

4. Refaire la même chose mais pour le bureau de l'utilisateur pour ajouter le raccourci vers l'application.

5. Idem pour le menu Démarrer/Programmes de l'utilisateur pour ajouter les dossiers et raccourcis vers l'application.

6. Lancer la construction du projet, un dossier `Release` contient alors le fichier .msi à déployer.

16. Master.HelpDesk

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Master.HelpDesk	20 mn	Cours, travaux dirigés	Individuel ou par binome
Contrôle des mise à jour, compilation, contrôle du style et des commentaires.			

Contrôle des mise à jour, compilation, contrôle du style et des commentaires.

Tests effectués. Suite des travaux.

17. Découverte d'un nouvel outil de développement

Déroulement

Phase	Durée	Type de travail	Modalités de travail
Découverte d'un nouvel outil de développement	15 mn	Présentation, modalités d'installation et essais	Individuel
Découverte de la documentation des sources, Ghostdoc, génération automatique de la documentation technique avec SandCastle.			

La documentation est un élément primordial dans le développement logiciel. Visual Studio dispose de tous les outils pour préparer les balises documentaires et générer les fichiers XML qui serviront à générer la documentation technique des classes. GhostDoc est un outil de génération des balises à partir du code et par une seule commande de touches. Chaque projet doit avoir la case "Fichier de documentation XML" cochée dans l'écran propriétés.

Télécharger les logiciels ci-dessous pour générer les fichiers .CHM de documentation technique :

Le programme SandCastle : SandCastle [http://www.microsoft.com/downloads/details.aspx?FamilyId=E82EA71D-DA89-42EE-A715-696E3A4873B2]

L'interface de paramétrage d'un projet SandCastle : SandCastle Help File Builder [http://www.codeplex.com/SHFB]

Le compilateur de documentation CHM de MicroSoft : HTML Help Workshop [http://msdn2.microsoft.com/en-us/library/ms669985.aspx]