025914 - Assurance de qualité logicielle Évaluation formative 2.1

Évaluation formative 2.1

# Objectif

Se familiariser avec les messages de diagnostiques les plus fréquemment produits lors de la compilation d'un code source : erreurs de syntaxe, avertissements (warnings) et erreurs de sémantique.

# Problème

- Donner 5 exemples de codes avec des erreurs de syntaxe

|  |  |  |
| --- | --- | --- |
| Code c# | Erreur ou message de synthaxe | solution |
| public class Livre  {  static public void Main()  {  }  }  public class Livre  {  public static void Main() // CS0017, delete one Main or use /main  {  }  } | Déclaration de 2 mêmes classes et 2 mêmes fonctions | public class Livre  {  static public void Main()  {  }  } |
| public class b  {  public static void Main()  {  1/0  }  } | Impossible de faire l’opération 1/0, syntaxiquement incorrecte. | public class b  {  public static void Main()  {  }  } |
| public class Program  {  public static void Main(string[] args)  {  int number = 5;  if (number > 0  {  Console.WriteLine("Number is positive");  }  }  } | CS1026 : ) attendue à if (number > 0 | public class Program  {  public static void Main(string[] args)  {  int number = 5;  if (number > 0)  {  Console.WriteLine("Number is positive");  }  }  } |
| public class Program  {  public static void Main(string[] args)  {  int[] numbers = new int[3];  numbers[0] = 1;  numbers[1] = 2;  numbers[2] = 3;  numbers[3] = 4;  }  } | L’affichage de ce code est incorrect, il est sensé affiché 3 nombres en commençant par 0 et finissant par 2 et non 3 | public class Program  {  public static void Main(string[] args)  {  int[] numbers = new int[4];  numbers[0] = 1;  numbers[1] = 2;  numbers[2] = 3;  numbers[3] = 4;  }  } |
|  |  |  |

025914 – Assurance de qualité logicielle

-Donner 5 exemples de codes avec des erreurs d’avertissement (warnings)

|  |  |  |
| --- | --- | --- |
| Code c# | Erreur ou message de synthaxe | solution |
| class MyClass  {  public static void Main()  {  int i;  for (i = 0; i < 10; i+= 1) ;  // CS0642 semicolon intentional?  {  System.Console.WriteLine(i);  }  }  } | Possibilité d’instruction vide erronée | class MyClass  {  public static void Main()  {  int i;  for (i = 0; i < 10; i ++);  System.Console.WriteLine(i);  }  }  } |
| class C  {  private int i = 1;  public static void Main()  { }  } | CS0414 : déclaration d’une variable non utilisée | class C  {  private int i = 1;  public static void Main()  {  System.Console.WriteLine(i);  }  } |
| class Test // CS0661  {  public static bool operator ==(object o, Test t)  {  return true;  }  public static bool operator !=(object o, Test t)  {  return true;  }  public override bool Equals(object o)  {  return true;  }  public static void Main()  {  }  } | CS0659 : Program.Test se substitue à  Object.Equals(object o) mais pas à Object.Get.HashCode()  CS0661 : Program.test définit l’opérateur == ou l’opérateur != mais ne se substittue pas à Object.Get.HashCode() | Private class Test  {  public static bool operator ==(object o, Test t)  {  return true;  }  public static bool operator !=(object o, Test t)  {  return true;  }  public override bool Equals(object o)  {  return true;  }  public static void Main()  {  }  }  CS0659 |
| class MyClass  {  public static void Main()  {  Int i = 0;  if (i < null) ;  i++;  }  } | CS0464 : La comparaison avec null de type int produit toujours false  CS0642 : Possibilité d’instruction vide érronée | class MyClass  {  public static void Main()  {  int? i = 0;  if (i >0)  i++;  }  } |
| public class Test  {  public static int Main()  {  int i = 5;  int counter = 0;    if (i == null)    counter++;  return counter;  }  } | CS0472: une Valeur de type int n’est jamais = null de type int | public class Test  {  public static int Main()  {  int i = 5;  int counter = 0;    if (i <0)    counter++;  return counter;  }  } |

-Donner 5 exemples de codes avec des erreurs de sémantique

|  |  |  |
| --- | --- | --- |
| Code c# | Erreur ou message de sémantique | solution |
| public class Program  {  public static void Main(string[] args)  {  int[] numbers = { 1, 2, 3, 4, 5 };  for (int i = 0; i <= numbers.Length; i++)  {  Console.WriteLine(numbers[i]);  }  }  } | La boucle for avec l’option « .length » compte lui-même le nombre d’itteration.  Pas besoin de mettre <= encore | public class Program  {  public static void Main(string[] args)  {  int[] numbers = { 1, 2, 3, 4, 5 };  for (int i = 0; i < numbers.Length; i++)  {  Console.WriteLine(numbers[i]);  }  }  } |
| public class Program  {  public static void Main(string[] args)  {  int a = 5;  int b = 0;  int result = a / b;  Console.WriteLine("Result: " + result);  }  } | Le code est bien écrit mais sémantiquement n’est pas sensé. Un nombre divisé par 0 est impossible | public class Program  {  public static void Main(string[] args)  {  int a = 5;  int b = 0;  if (b != 0)  {  int result = a / b;  Console.WriteLine("Result: " + result);  }  else  {  Console.WriteLine("Cannot divide by zero");  }  } |
| public class Program  {  public static void Main(string[] args)  {  int result = Add(5, 3);  Console.WriteLine("Result: " + result);  }  public static int Add(int a, int b)  {  return a - b;  }  } | On ne peut déclarer vouloir faire l’addition et par la suite retourner la soustraction plutôt | public class Program  {  public static void Main(string[] args)  {  int result = Add(5, 3);  Console.WriteLine("Result: " + result);  }  public static int Add(int a, int b)  {  return a +b;  }  } |
| public class Program  {  public static void Main(string[] args)  {  int number = 10;  bool isPositive = CheckIfPositive(number);  Console.WriteLine("Is Positive: " + isPositive);  }  public static bool CheckIfPositive(int number)  {  return number < 0;  }  } | Le code a pour but précis de savoir si le nombre est positif, pas de retourner un nombre négatif. | public class Program  {  public static void Main(string[] args)  {  int number = 10;  bool isPositive = CheckIfPositive(number);  Console.WriteLine("Is Positive: " + isPositive);  }  public static bool CheckIfPositive(int number)  {  return number > 0;  }  } |
|  |  |  |

# Matériel fourni

S’inspirer des exemples de la présentation du cours disponible sur eCité : « Stratégies de débogage\_erreurs de syntaxe et de sémantique. .pptx »

# À soumettre

Une fois l’exercice solutionné, démontrez à l’instructrice que ça fonctionne bien. Puis soumettez votre solution dans le pigeonnier correspondant sur eCité.

# Suggestions

Référez-vous au site de Microsoft portant sur les erreurs de compilation de C# avec l’IDE Visual Studio Community:

<https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/compiler-messages/>

©Manel Sorba 2024 Page EF2.1 Collège La Cité