# Calcul du PGCD

**Objectifs** : se familiariser avec la syntaxe C#, les opérateurs et structures de contrôle, la saisie et l’affichage sur la console

On souhaite écrire un programme de calcul du pgcd de deux entiers non nuls, en C# à partir de l’algorithme de la méthode dite « égyptienne ». Voici une spécification de l'algorithme de calcul du PGCD de deux nombres (entiers strictement positifs) p et q, selon cette méthode :

Lire (p, q);

Tantque p ≠ q faire

Si p > q alors

p ← p – q

sinon

q ← q – p

FinSi

FinTant;

Ecrire(" PGCD = ", p)

Ecrivez le programme C# complet qui produise le résultat suivant :

Entrez le premier nombre : 21

Entrez le deuxième nombre : 45

Le PGCD de 21 et 45 est : 3

# Calcul de nombres premiers

**Objectifs** : se familiariser avec la syntaxe C#, les opérateurs et structures de contrôle, la saisie et l’affichage sur la console.

On souhaite écrire un programme C# de calcul et d'affichage des N premiers nombres premiers. Un nombre entier est premier s’il n’est divisible que par 1 et par lui-même (ex : 1, 2, 3, 5, 7, 11, 13…)

Le programme ne doit utiliser que des boucles (do) while, mais des boucles for.

Spécifications de l’algorithme : (on étudie la primalité des nombres uniquement impairs)

Algorithme Premier

Entrée: n ∈ N

Sortie: nbr ∈ N

Local: estPremier ∈ {Vrai , Faux}

divis, compt ∈ N²;

début

lire(n);

compt ← 1;

ecrire(2);

nbr ← 3;

Tantque(compt < n) Faire

divis ← 3;

Est\_premier ← Vrai;

Répéter

Si reste(nbr par divis) = 0 Alors

estPremier ← Faux

Sinon

divis ← divis+2

Fsi

jusquà (divis > nbr / 2) ou (estPremier=Faux);

Si estPremier = Vrai Alors

ecrire(nbr);

compt ← compt+1

Fsi;

nbr ← nbr+2 // nbr impairs

Ftant

FinPremier

Ecrivez le programme C# complet qui produise le résultat suivant :

Combien de nombres premiers : 5

2

3

5

7

11

# Tri d’un tableau

**Etape 1** : Créer une fonction permettant d’afficher le contenu d’un tableau (sur une seule ligne).

**Etape 2** : Créer une fonction permettant de trier par ordre alphabétique les éléments d’un tableau de mots passé en paramètre.

Indications :

* Pour trier un tableau, on peut comparer ses éléments deux à deux, et les permuter jusqu’à ce qu’on arrive à parcourir le tableau sans faire aucune permutation.
* Pour comparer une chaîne avec une autre, on peut utiliser la méthode CompareTo(). Exemple : mot1.CompareTo(mot2) ;

**Etape 3** : tester le tri grâce à la première fonction

**Etape 4** : modifier le type du paramètre d’entrée de la fonction en IComparable[]. Que constatez-vous ?

**Etape 5** : faire en sorte que la fonction ne modifie pas le tableau passé en paramètre, mais renvoie plutôt le tableau trié en résultat. Tester avec la fonction d’affichage.

# Comptage des voyelles et consonnes

**Objectifs** : se familiariser avec les bases du langage C# : tableaux, fonctions, passage de paramètres en ref et out

**Etape 1** : Dans la fonction Main() :

* Faire saisir un mot à l’utilisateur. On ne fera pas de vérification du mot saisi ; on s’attend à ce qu’il ne comporte que des lettres.
* Créer une fonction vide qui prend en entrée un mot, calcule ses nombres de voyelles et consonnes, et les renvoie en paramètres out.
* Afficher le résultat de l’appel de cette fonction sous la forme : « ”livre” comprend 3 consonnes et 2 voyelles »

**Etape 2** : implémenter le corps de la fonction vide créée précédemment et tester.

# Login

**Objectifs** : apprendre à émettre et intercepter des exceptions

**Etape 1** : Pour créer un compte, demander à l’utilisateur de saisir successivement un login puis un mot de passe. Afficher ensuite un message « Votre compte a bien été créé. Un message vient de vous être envoyé »

**Etape 2** : Créer 2 fonctions pour vérifier les formats des informations du compte :

* Le login doit faire au moins 5 caractères.
* Le mot de passe doit comporter au moins 6 caractères, au plus 12, et ne doit pas commencer ni finir par un espace.

Si les formats ne sont pas bons, lever des exceptions du type FormatException, avec des descriptions explicites.

**Etape 3** : Dans le code de l’étape 1, appeler la méthode de vérification du login. Intercepter l’exception sur le format et afficher le message correspondant.

**Etape 4** : Faire en sorte que la demande de login soit répétée tant qu’un login correct n’a pas été saisi

**Etape 5** : appliquer les 2 dernières étapes à la gestion du mot de passe. Le mot de passe ne doit être demandé que si la saisie du login est correcte.

Boîtes

**Objectifs** : mettre en œuvre les notions suivantes : constantes, propriétés, méthodes, constructeurs, surcharges et agrégation. Et de façon annexe : énumérations, exceptions et structuration du code.

**Etape 1** : Dans un nouveau projet nommé Boites, créer une classe nommée Boite avec des propriétés en lecture seule pour sa hauteur, sa largeur, sa longueur, toutes 3 initialisées par défaut à 30.0.

**Etape 2** : Créer un type énuméré pour les couleurs (blanc, bleu, vert, jaune, orange, rouge, marron). Créer une propriété en lecture/écriture de ce type sur la classe Boite.

**Etape 3** : Créer une propriété énumérée en lecture seule pour la matière (carton, plastique, bois, métal), initialisée par défaut à Carton.

**Etape 4** : Créer une propriété Volume en lecture seule qui retourne le volume calculé d’après les dimensions.

**Etape 5** : Ajouter une méthode publique Etiqueter, avec un paramètre « destinataire » de type string. Créer une surcharge avec un paramètre booléen supplémentaire nommé « fragile ». La surcharge doit appeler la première méthode Etiqueter.

**Etape 6** : Ajouter une méthode Compare qui renvoie vrai si les dimensions et la matière des 2 boîtes sont identiques. La tester.

**Etape 7** : Ajouter 2 constructeurs à la classe Boîte. Le 1er permet d’initialiser ses dimensions. Le 2d permet d’initialiser en plus sa matière. Tester ces constructeurs en créant des boîtes dans la fonction Main

**Etape 8** : Faire en sorte que le second constructeur appelle le premier

**Etape 9** : Ajouter une classe Etiquette avec 3 propriétés en lecture/écriture : Texte (string), Couleur (énuméré déjà créé), Format (énuméré avec valeurs XS, S, M, L, XL)

**Etape 10** : ajouter un compteur d’instances sur la classe Boite, en utilisant une propriété statique pour retourner sa valeur. Tester son fonctionnement.

**Etape 11** : Dans la classe Boite, ajouter 2 champs privés de type Etiquette. Nommez-les \_etiquetteDest, \_etiquetteFragile

Dans la première méthode Etiqueter créée à l’étape 5, créer une instance d’Etiquette de couleur blanche, de format L, et dont le texte est le destinataire. Cette instance doit être accessible via le champ \_etiquetteDest

Dans la seconde méthode Etiqueter créée à l’étape 5, si le paramètre fragile vaut Vrai, créer une instance d’étiquette de couleur rouge, de format S, et dont le texte vaut « FRAGILE ». Cette instance doit être accessible via le champ \_etiquetteFragile.

Dans la fonction Main, créer une boîte, et l’étiqueter avec son destinataire et une étiquette « FRAGILE ».

On vient d’illustrer la composition.

**Etape 12 :** Créer une 3ème surcharge de la méthode Etiqueter, qui prend deux paramètres de type Etiquette (une pour le destinataire et une pour indiquer le caractère fragile)

 Les étiquettes passées en paramètre doivent être affectées aux champs \_etiquetteDest et \_etiquetteFragile

Dans la méthode Main de la classe Program

* En utilisant le constructeur adéquat, créer une boîte en plastique de dimensions 30 x 40 x 50
* En utilisant un initialiseur, créer une étiquette de couleur blanche, de format L, avec comme texte un destinataire de votre choix
* Créer de la même façon une étiquette rouge de format S, avec le texte « FRAGILE »
* Affecter ces étiquettes à la boîte à l’aide de la méthode Etiqueter créée précédemment

On vient d’illustrer l’agrégation.

Véhicules

**Objectifs** : mettre en pratique les notions suivantes : héritage, appels des constructeurs, classes abstraites, implémentation d’interface

**Etape 1** : Créer un type énuméré nommé Energies avec les valeurs : Aucune, Essence, Gazole, GPL, Electrique

Créer une classe Véhicule avec 3 propriétés en lecture : Nom, NbRoues et Energie

Ajouter un constructeur pour initialiser ces propriétés.

**Etape 2** : Créer une classe dérivée Voiture

Générer le constructeur proposé par VS, qui appelle celui de Véhicule

Mettre la valeur 4 pour le paramètre du nombre de roues.

Tester en instanciant une voiture et en exécutant le code pas à pas en debug.

Noter qu’on n’a pas eu besoin de propriétés en écriture.

**Etape 3** : Dans la classe Véhicule, ajouter une propriété virtuelle Description qui renvoie une chaîne « Véhicule X roule sur X roues et à l’énergie X » (remplacer X par les valeurs de propriétés).

Dans Voiture, redéfinir cette propriété. Par défaut VS génère automatiquement le corps en ajoutant un appel à la propriété de la classe ancêtre. Ajouter « Je suis une voiture \r\n » devant l’appel.

Dans Main, ajouter l’affichage de la description de la voiture et exécuter.

**Etape 4** : Créer une classe Moto dérivée de Véhicule et redéfinir la propriété Description

Dans la fonction Main, créer une instance de Voiture et une instance de Moto en les référençant par des variables de type Véhicule. Afficher la valeur de la propriété Description à partir de ces 2 variables.

On obtient un comportement polymorphique

**Etape 5** : rendre la classe Véhicule abstraite et ajouter une méthode abstraite void CalculerConso

Noter que VS souligne Voiture et Moto en rouge pour signaler que la méthode CalculerConso n’est pas implémentée.

Cliquer sur le nom puis sur l’info-bulle pour implémenter cette méthode dans chaque classe

Dans Main, essayer d’instancier Véhicule. VS souligne la ligne en rouge car cette classe est abstraite.

**Etape 6** : Ajouter une propriété PRK (Prix de revient kilométrique) abstraite en lecture seule sur Véhicule

Fournir une implémentation dans les classes dérivées (renvoyer simplement une valeur arbitraire)