



Département d'informatique

**Logique de programmation
420-1A5-LL**

Cahier d'apprentissage

**Par Christian Asselin
Département d'Informatique
Révision juin 2020**

Session Automne 2020

RÈGLES DE PROGRAMMATION pour le cours Logique de programmation

*Un programme qui MARCHE n'est PAS NÉCESSAIREMENT
un BON programme !!!*

En plus de produire des résultats exacts, ce qui est la moindre des choses, un BON programme doit comporter les qualités suivantes:

- Bonne lisibilité générale
- Efficacité minimale...au moins!
- Respect des normes d'organisation

Des points pourront être enlevés en tout temps pour un programme de qualité insuffisante.

Normes obligatoires pour le cours Logique de programmation (420-1A5-LL):

- *Entête obligatoire: nom de l'étudiant et nom du programme;*
- *Règles du Si (complexe vs imbriqué vs successif) et du switch case telles qu'appriées en Logique de programmation (420-1A5-LL);*
- *Décalages significatifs;*
- *Utilisation du Pour pour des boucles avec n défini et du Tant Que pour des boucles avec n indéfini;*
- *Pour des boucles indéfinies, la seule forme acceptable est le Tant Que; l'utilisation d'un Pour avec une instruction de cassure (break ou autre) ou de modification de la variable de test est inacceptable;*
- *Termes français pour tous les affichages (message et écran);*
- *JAMAIS de ligne de plus de 80 caractères;*
- *Une seule instruction par ligne;*
- *Tout commentaire doit être mise-à-jour en tout temps;*
- *AUCUNE variable globale (sauf exception justifiée par le professeur, et alors préfixée par g_);*
- *TOUTES les variables membres doivent être préfixées par m_;*
- *Noms de variable et de méthode/fonction significatifs et comportant au besoin d'une lettre majuscule au début de chacun des noms qui le compose (i.e. Notation Pascal); ex : SalaireEmploye*
- *Méthodes/fonctions de 40 lignes maximum;*
- *copier/coller doit être découragé au profit de l'utilisation de méthodes/fonction;*

Contenu

1	Introduction à l'approche algorithmique	1-7
1.1	Le concept algorithmique	1-8
1.2	Les structures de base	1-9
1.3	Des petits exercices	1-9
2	Processus de résolution d'un problème	2-10
2.1	Les étapes de résolution d'un problème	2-11
2.2	Un exemple des étapes (1 à 5) de conception sur papier	2-12
2.3	L'algorithme	2-13
2.3.1	La constante	2-13
2.3.2	La variable	2-14
2.3.3	Les Mots Réservés	2-14
2.3.4	L'instruction	2-14
2.3.5	L'instruction d'affectation	2-14
2.3.6	Les opérateurs arithmétiques	2-15
2.3.7	La priorité des opérateurs	2-15
2.3.8	L'instruction Lis	2-16
2.3.9	L'instruction Écris	2-16
2.3.10	L'algorithme	2-17
2.3.11	La trace	2-17
2.4	Un exemple d'algorithme complet	2-18
2.5	Le quiz de révision	2-19
3	Introduction au langage C#	3-25
3.1	La conception visuelle avec Microsoft Visual C#.Net	3-26
3.1.1	Convention pour l'attribution du nom des contrôles	3-27
3.1.2	La table des propriétés	3-28
3.2	Les commentaires et règles d'écriture	3-30
3.3	Les variables	3-30
3.3.1	L'attribution de noms aux variables	3-30
3.4	Les types de données	3-31
3.4.1	La conversion entre les types numériques :	3-32
3.5	L'entête de la méthode	3-34
3.6	La déclaration des variables locales à une méthode	3-34
3.7	La traduction de l'instruction algorithmique Lis	3-35
3.8	La traduction de l'instruction algorithmique d'affectation	3-36
3.9	La traduction de l'instruction algorithmique Écris	3-36
3.10	Une méthode complète en C#	3-37
3.11	Le quiz de révision	3-39

3.12	Le résumé et le vocabulaire	3-43
4	Structure alternative	4-44
4.1	Le format général	4-46
4.2	Des exemples	4-47
4.3	L'expression simple	4-48
4.4	L'expression complexe	4-48
4.5	L'évaluation des expressions complexes	4-48
4.6	Le quiz de révision	4-49
4.7	La traduction de l'instruction Si en C#	4-51
4.7.1	Les règles d'écriture	4-51
4.7.2	L'expression	4-52
4.7.3	Les opérateurs relationnels	4-52
4.7.4	Les opérateurs logiques	4-52
4.8	La validation des entrées	4-53
4.9	La formule de l'arrondi	4-60
4.10	Le quiz de révision	4-60
5	Compteur, accumulateur et variable membre	5-65
5.1	Les compteurs et les accumulateurs	5-66
5.1.1	Les accumulateurs	5-66
5.1.2	Les compteurs	5-66
5.2	La variable locale	5-67
5.3	La variable membre d'une classe	5-67
5.4	Un exemple nécessitant des compteurs et des accumulateurs	5-68
5.4.1	L'analyse:	5-69
5.4.2	La conception visuelle de l'application	5-69
5.4.3	Les variables membres nécessaires	5-70
5.4.4	L'algorithme Constructeur	5-71
5.4.5	L'algorithme du bouton Emission	5-71
5.4.6	L'algorithme du bouton Total des ventes	5-72
5.4.7	La traduction en C#	5-72
5.5	Une nouvelle classe d'un formulaire dans un projet Visual C#	5-76
5.6	Le quiz de révision	5-80
6	Structure alternatives simple, multiple et imbriquée	6-81
6.1	L'utilisation d'une expression simple et multiple	6-82
6.2	Le Si imbriqué	6-82
6.3	L'utilisation des Si imbriqués	6-83
6.4	Les étapes à suivre pour résoudre une structure alternative	6-85
6.5	Le quiz de révision	6-85
		1-4

7	Structure répétitive	7-90
7.1	Définition	7-91
7.2	Des exemples tirés de la vie courante	7-91
7.3	La structure répétitive Tant que	7-92
7.4	La traduction en C# du Tant Que	7-93
7.5	La structure répétitive Faire...Tant que	7-93
7.6	La traduction en C# du Faire...Tant Que	7-94
7.7	La structure répétitive Pour	7-95
7.8	Traduction en C# du Pour	7-96
7.9	Des exemples	7-97
7.10	Les boucles imbriquées	7-98
7.11	Le quiz de révision	7-99
8	Tableaux	8-102
8.1	Définition	8-103
8.2	Les tableaux à une dimension	8-103
8.2.1	Cas d'un tableau de caractères	8-104
8.2.2	Cas d'un tableau de chaînes	8-105
8.3	La traduction en C# des tableaux à une dimension	8-105
8.3.1	La déclaration	8-105
8.3.2	L'affectation	8-105
8.3.3	L'accès à un élément	8-105
8.4	Les algorithmes de base de manipulation de tableaux	8-106
	Particularité du C# et son environnement de développement .NET	8-108
8.5	L'instruction SWITCH-CASE	8-109
8.6	L'utilisation des contrôles Cases à Cocher	8-111
8.7	L'utilisation des contrôles Boutons Radio	8-113
8.8	L'utilisation des ComboBox	8-115
8.9	L'affichage dans une boucle	8-116
8.9.1	Affichage dans une boucle en utilisant plusieurs MessageBox	8-116
8.9.2	L'affichage dans une boucle en utilisant un seul MessageBox	8-116
8.9.3	L'affichage dans une boucle en utilisant un ListBox	8-117
8.10	La classe string	8-118
8.10.1	Définition	8-118
8.10.2	Les opérateurs d'égalités	8-118
8.10.3	L'opérateur de concaténation (+)	8-118
8.10.4	La propriété Length	8-118
8.10.5	Accéder à un caractère d'une chaîne	8-119
8.10.6	Les méthodes de base de la classe string	8-119

8.10.7	Les autres méthodes de la classe string	8-120
8.11	La structure char	8-121
8.11.1	Des exemples d'utilisation	8-121
8.12	Des exercices	8-122
8.13	Les raccourcis d'expression arithmétiques et les méthodes mathématiques	8-124
8.13.1	Les opérateurs équivalents en C#	8-124
8.13.2	La classe Math	8-124
8.13.3	Des exemples d'utilisation	8-125
9	Quelques notions de graphisme en C#	9-126
9.1	Le graphique GDI+	9-127
9.1.1	La création d'un objet Graphics	9-127
9.1.2	Le dessin et la manipulation de formes et d'images	9-128
9.2	Les stylets, brosses et couleurs	9-128
9.2.1	Les stylets	9-128
9.2.2	Les brosses	9-129
9.2.3	Les couleurs	9-130
9.3	Le dessin de traits et de formes avec GDI+	9-131
9.4	Le dessin de texte avec GDI+	9-132
9.5	Le rendu d'images avec GDI+	9-132
10	Exercices synthèses	10-134
10.1	Les Si simples, imbriqués et multiples	10-135
10.2	La validation de chaînes de caractères	10-135
10.3	Les boucles simples	10-136
11	Travail préparatoire obligatoire en laboratoire.	11-137
12	Travaux pratiques	12-158
12.1	Travail Pratique No. 1 : Les propriétés des contrôles	12-159
12.2	Travail Pratique No. 2 : La structure séquentielle	12-167
12.3	Travail Pratique No. 3 : Les structures séquentielles et alternatives	12-169
12.4	Travail Pratique No. 4 : Les structures séquentielles et alternatives	12-171
12.5	Travail Pratique No. 5 : Les structures séquentielles et alternatives	12-173
12.6	Travail Pratique No. 6 : Les structures séquentielles, alternatives et répétitives	12-176
12.7	Travail Pratique No. 7 : Les Tableaux	12-179
12.8	Travail Pratique No. 8 : La structure répétitive	12-185
12.9	Travail Pratique No. 9 : Les Boucles imbriquées et les méthodes de dessin	12-188
12.10	Travail pratique No. 10 : Les compteurs et accumulateurs.	12-190
13	Annexe 1 – Table des codes ASCII	13-192

1 Introduction à l'approche algorithmique

Lorsque l'homme effectue un calcul ou un travail quelconque, il n'est pas tenu de décomposer toutes les étapes de son raisonnement. Mais si ce travail doit être exécuté par ordinateur il faut tout décomposer.

Ex.: Classer une liste de 3 noms par ordre alphabétique.

1.1 Le concept algorithmique

- 1) L'algorithme est la description des étapes à suivre pour atteindre les résultats et objectifs visés.
- 2) L'algorithme est la décomposition d'un problème en plusieurs actions (très détaillées).
- 3) Les actions sont écrites en français de façon indépendante du langage informatique utilisé.
- 4) La possibilité d'avoir plusieurs algorithmes pour résoudre une même situation problématique.

On a la forme d'algorithmes:

Textuel (**pseudo-code**)

Problème : Donner le plus petit de deux nombre lue au clavier.

Lis Nombre1, Nombre2

Si Nombre1 < Nombre2

PlusPetit ← Nombre1

Sinon

PlusPetit ← Nombre2

Écris PlusPetit

Lis (les données d'entrées qu'on va avoir besoin pour faire un traitement)

Le symbole d'affectation ← (on va porter la nouvelle valeur de droite vers la gauche dans un récipiant qui est une variable)

Écris (On affiche la ou les donnée(s) de sortie, i.e. la ou les valeur(s) du résultat du traitement)

1.2 Les structures de base

En programmation structurée, pour construire des algorithmes, on utilise trois (3) structures de base:

Séquentielle, alternative et répétitive

En fait, un algorithme est la description détaillée des **étapes logiques** pour résoudre un problème. Il sert de base à la construction d'un programme informatique qui lui est écrit dans un langage de programmation assimilable par l'ordinateur. (C++, JAVA, PHP, PYTHON etc....)

Durant cette session, les algorithmes seront traduits en **C# (prononcez C Sharp)**. Une fois saisi dans un fichier, notre programme en **C#** doit être compilé c'est-à-dire que le logiciel de programmation (le compilateur) **traduit chacune des lignes de code en langage machine et détecte les erreurs de syntaxe**.

Le résultat obtenu s'appelle le **programme objet**. Une fois toutes les erreurs corrigées, **l'éditeur des liens rassemble** tous les outils nécessaires à l'exécution de notre programme en un fichier **exécutable (fichier à extension exe)**. C'est notre application.

1.3 Des petits exercices

a) Nous avons l'algorithme suivant :

```
Lis Nombre1, Nombre2
Si Nombre1 < Nombre2
    PlusPetit ← Nombre1
Sinon
    PlusPetit ← Nombre2
Écris PlusPetit
```

Quel sera l'affichage si les valeurs 10 et 20 sont lues ? _____
Quel sera l'affichage si les valeurs 30 et 15 sont lues ? _____
Quel sera l'affichage si les valeurs 30 et 30 sont lues ? _____

b) Nous avons l'algorithme suivant :

```
Prix ← 25,50
Quantite ← 5
Rabais ← 0,10
TotalBrut ← Prix * Quantite
TotalRabais ← TotalBrut * Rabais
TotalNet ← TotalBrut – TotalRabais
Écris TotalBrut, TotalRabais, TotalNet
```

Quel sera l'affichage ? _____

2 Processus de résolution d'un problème

2.1 Les étapes de résolution d'un problème

On distingue plusieurs grandes étapes dans le processus de résolution d'un problème.

1) La réception de l'énoncé du problème.

Comprendre exactement ce qui est demandé ; *le problème à résoudre*.

2) La compréhension des données du problème.

Identifier les *données d'entrée* et les *données de sortie*. Est-il possible avec les données d'entrée dont on dispose, d'en arriver à trouver les sorties ? Sinon, on a besoin de retourner à l'étape 1.

3) La conception visuelle de l'application.

- De quoi sera composée la **fenêtre produite** par mon programme ?
- Comment seront disposées les données (**entrées et sorties**)?
- Comment l'utilisateur utilisera mon programme ?

4) La construction de l'algorithme ou des algorithmes.

- Identifier les **opérations mathématiques** ou de **classements** ou **autres** nécessaires pour passer des données d'entrée aux données de sortie;
- **Organiser la séquence des traitements** à effectuer pour **arriver aux résultats souhaités**.

5) La vérification de l'algorithme avec des valeurs (essai/trace de l'algorithme).

6) La conception visuelle de l'application sur ordinateur avec Microsoft Visual C# en mode design.

7) La traduction de l'algorithme en C# et association du traitement avec un contrôle de notre application.

8) La vérification du programme avec les mêmes données d'essais qui ont servi à vérifier l'algorithme, celles du point 5

Note : Les étapes 1 à 5 se font sur papier sans ordinateur.

Les étapes 6,7 et 8 se font à l'ordinateur.

À chacune des étapes, il peut être nécessaire de retourner aux étapes précédentes.

2.2 Un exemple des étapes (1 à 5) de conception sur papier

L'énoncé du problème : **Calcul de paie simplifié.**

À partir du nom d'une personne, de son salaire horaire et du nombre d'heures travaillées par cette personne dans une semaine, on voudrait connaître son salaire brut ainsi que son salaire net sachant que l'on prélève 25% du salaire brut en impôt. Après avoir saisi les données nécessaires, on appuiera sur un bouton pour obtenir les résultats souhaités.

La compréhension des données:

<u>ENTRÉES</u>	<u>SORTIES</u>
Nom	SalaireBrut
TauxHoraire	SalaireNet
NombreHeures	

La conception visuelle :

Calcul de paie simplifié

Nom

Taux horaire

Nombre d'heures

Salaire brut

Salaire net

L'algorithme du bouton Calculer: **Lis** Nom, TauxHoraire, NombreHeures

SalaireBrut \leftarrow TauxHoraire * NombreHeures

Impot \leftarrow SalaireBrut * 25 / 100

SalaireNet \leftarrow SalaireBrut - Impot

Écris SalaireBrut, SalaireNet

La trace d'un jeu d'essais :

Variable	Essai 1	Écran
Nom	"Pierre"	
TauxHoraire	15	
NombreHeure	20	
SalaireBrut	300	
Impot	75	
SalaireNet	225	

2.3 L'algorithme

2.3.1 La constante

Valeur fixe, qui ne change jamais dans le temps ou durant toute la durée de l'application. On l'utilise dans les instructions algorithmiques.

On distingue trois (3) **grandes catégories** de constantes:

Les constantes numériques :

Elles représentent **les nombres** et peuvent intervenir dans les calculs arithmétiques.

Elles sont divisées en **deux (2) sous-catégories**:

Les constantes numériques entières: Ce sont les nombres entiers positifs, négatifs ou nuls.

Exemple: 12 -56 0

Les constantes numériques réelles: Ce sont les nombres avec une partie décimale. Ceux-ci peuvent être positifs, négatifs, ou nuls.

Exemples: 34,987 -72,5 0,0

Les constantes alphanumériques:

Elles représentent les caractères et ne peuvent pas intervenir dans les calculs arithmétiques.

Elles sont divisées en deux (2) sous-catégories:

Les constantes caractères: Un seul caractère que l'on place entre *apostrophes*.

Exemple: 'A' 'b' '5' '-'

Les constantes chaînes de caractères: Une suite de caractères que l'on place entre **guillemets**.

Exemple: "Christian" "833-5110" "Assc2019"

Il est très important de faire la distinction entre **un caractère** et **une chaîne** de caractères. Pour l'ordinateur se sont deux concepts très différents. La distinction se fait en utilisant soit les **apostrophes** (**caractère**) soit les **guillemets** (**chaîne**).

Les constantes booléennes :

Elles représentent soit la valeur **VRAI** ou la valeur **FAUX**.

2.3.2 La variable

C'est un **nom symbolique** qui représente une certaine valeur qui peut changer dans le temps.

La valeur de la variable est conservée en mémoire et on y accède par son **nom**.

La variable est un **contenant** dont le **contenu** peut changer suivant l'évolution de notre algorithme. On peut faire l'analogie avec un tiroir à ustensiles et les ustensiles qu'il contient : ici le tiroir est l'espace mémoire dans lequel on dépose les ustensiles qui y sont déposés et retirés selon les besoins.

Pour nommer **une variable**, on utilisera un ou des mots significatifs (**au long** ou **abréviation significative**); **la première lettre de chaque mot est une majuscule; pas d'espace entre les mots, ni d'accent; chaque mot écrit au singulier.**

La variable peut être **numérique**, **alphanumérique** ou **booléenne** selon le **type de valeur** qu'elle contiendra à sa première utilisation dans l'algorithme.

Toujours à partir de l'analogie avec le tiroir, celui qui contient les ustensiles s'appelle le tiroir à ustensiles et celui qui contient les linges à vaisselle s'appelle le tiroir à linges à vaisselle.

**Exemples: TiroirUstensile TiroirLingeVaisselle Nom Prenom NomDeFamille
SalaireBrut SalBrut Code**

À ne pas faire : Prénom (pas d'accent) Lesalaire (chaque mot commence par une majuscule)

2.3.3 Les Mots Réservés

Mot ou symbole qui représente une action algorithmique. Ces mots ne doivent jamais servir à nommer une variable. Ces mots sont réservés à l'écriture des instructions composant l'algorithme.

Exemples: **Écris Lis ← Si Sinon Faire TantQue Pour**

2.3.4 L'instruction

Ligne d'un algorithme décrivant une action complète à effectuer. Cette ligne contient obligatoirement au moins un mot réservé.

Exemples : **Lis Code**
Total ← Nombre1 + Nombre2

2.3.5 L'instruction d'affectation

L'affectation ou l'assignation est l'opération qui consiste à placer une valeur dans une variable, à donner une valeur à une variable ou à changer la valeur d'une variable. C'est l'action de mettre un ustensile dans un tiroir.



Format général: **Variable ← Expression**

Variable est le nom de la variable qui recevra la valeur de l'expression.

Expression peut être une variable, une constante ou des variables et constantes reliées par un opérateur. Le résultat de l'expression doit être du même type que la variable à affecter.

La première utilisation d'une variable **détermine son type**. En poursuivant l'analogie avec le tiroir à ustensiles, ce tiroir devient un tiroir à ustensiles du moment que l'on dépose le premier ustensile. Par la suite, on n'y déposera pas de linges à vaisselle.

On utilise l'instruction d'affectation \leftarrow pour **placer la valeur** dans une variable.

```
NomFamille  $\leftarrow$  "Asselin"  
Prenom  $\leftarrow$  "Christian"  
NomComple  $\leftarrow$  Prenom + ", " + NomFamille  
Montant  $\leftarrow$  12,458  
Rabais  $\leftarrow$  Montant * 0,25  
MontantFinal  $\leftarrow$  Montant – Rabais
```

Des erreurs d'affectation : **pas le bon type de données**

```
Prenom  $\leftarrow$  12  puisque Prenom est de type chaîne de caractères (voir plus haut)  
Rabais  $\leftarrow$  "12,45"  puisque Rabais est de type réel (voir plus haut)
```

2.3.6 Les opérateurs arithmétiques

Opérateur	Rôle	Exemple d'utilisation
+	Addition	2 + 3 vaut 5
-	Soustraction	3 – 1 vaut 2
*	Multiplification	2 * 3 vaut 6
/	Division	6 / 3 vaut 2
%	Modulo	13 % 4 vaut 1

N.B. Le % est un symbole réservé en algorithmie. Pour indiquer le pourcentage on indique la formule.

2.3.7 La priorité des opérateurs

Ordre de priorité	Opérateur
1	()
2	*, / , %
3	+, -

Les opérations arithmétiques sont **toujours résolues de gauche à droite** selon la priorité des opérateurs. Pour changer la priorité, utiliser les parenthèses.

Exemples: $A \leftarrow 80 - 10 / 2$ affecte la valeur 75 à la variable A
 $A \leftarrow (80 - 10) / 2$ affecte la valeur 35 à la variable A

2.3.8 L'instruction Lis

L'instruction **Lis** permet d'affecter une valeur venant d'un périphérique d'entrée à une variable à l'exécution de l'algorithme. Les données à lire proviennent habituellement du clavier. Elles peuvent aussi provenir de fichiers ou d'autres périphériques d'entrée.

Donc, la lecture permet d'affecter une valeur aux variables d'entrée et cette valeur vient d'un périphérique.

Il y a toujours au moins une variable nommée suivant le mot réservé **Lis**. S'il y en a plus d'une, on sépare les variables à l'aide de la virgule.

Exemple: **Lis** Nom, Age, Salaire

2.3.9 L'instruction Écris

L'instruction **Écris** permet de transférer le contenu des variables vers un périphérique de sortie. Habituellement, on veut afficher le contenu des variables dans une boîte d'édition ou dans une boîte à message mais, on peut aussi vouloir les imprimer, les placer dans un fichier ou sur tout autre périphérique de sortie.

Il y a toujours au moins une variable nommée suivant le mot réservé **Écris**. S'il y en a plus d'une, on sépare les variables à l'aide de la virgule.

Exemples: **Écris** SalaireBrut, SalaireNet

2.3.10 L'algorithme

Un **algorithme** est une **suite d'instructions** (une instruction par ligne) composée de **mots réservés**, de **variables**, de **constantes** et de **opérateurs qui porte un nom**, ainsi que le nom du bouton.

On exécute un algorithme **instruction par instruction**, à partir de la **première instruction** jusqu'à la **dernière**. Un algorithme porte un **nom significatif**. C'est le traitement à faire.

Algorithme du bouton Salaire :

Lis TauxHoraire, NombreHeure

SalaireBrut \leftarrow TauxHoraire * NombreHeure

Impot \leftarrow SalaireBrut * 25 / 100

SalaireNet \leftarrow SalaireBrut – Impot

Écris SalaireBrut, SalaireNet

2.3.11 La trace

Pour vérifier la validité d'un algorithme, le seul moyen dont on dispose est de fixer des données d'entrée et de prévoir les résultats attendus. On exécute chacune des instructions de l'algorithme à partir de ces données d'entrée et on vérifie le résultat produit par l'algorithme avec les résultats attendus. C'est à celui qui écrit l'algorithme de simuler le travail de l'ordinateur.

Il est très important de simuler l'instruction telle qu'elle est écrite pour s'assurer du bon fonctionnement de l'instruction.

- Quelles seront les valeurs des variables A, B et C après l'exécution des instructions suivantes :

A \leftarrow 5

B \leftarrow 3

C \leftarrow A + B

A \leftarrow 2

C \leftarrow B – A

- Trace d'exécution:

Instruction	Contenu des variables		
	A	B	C
(1) A \leftarrow 5	5	-	-
(2) B \leftarrow 3	5	3	-
(3) C \leftarrow A + B	5	3	8
(4) A \leftarrow 2	2	3	8
(5) C \leftarrow B – A	2	3	1

2.4 Un exemple d'algorithme complet

L'algorithme du bouton Salaire :

Lis TauxHoraire, NombreHeure

SalaireBrut \leftarrow TauxHoraire * NombreHeure

Impot \leftarrow SalaireBrut * 25 / 100

SalaireNet \leftarrow SalaireBrut – Impot

Écris SalaireBrut, SalaireNet

Dans cet algorithme :

On lit deux (2) valeurs au clavier et elles sont affectées aux deux (2) variables : TauxHoraire et NombreHeure .

Il y a cinq (5) variables : TauxHoraire , NombreHeure, SalaireBrut, Impot et SalaireNet .

On écrit la valeur de deux (2) variables : SalaireBrut et SalaireNet.

Il y a une variable qui sert temporairement pour les calculs : Impot.

Il y a deux (2) constantes : 25 et 100.

Il y a trois opérateurs : la multiplication (*), la division (/) et la soustraction (-).

Il y a cinq (5) instructions : une instruction **Lis**, trois (3) instructions d'affectation et une instruction **Écris**.

Remarquer l'utilisation de la virgule pour séparer les variables dans les instructions **Lis** et **Écris**.

La trace d'un jeu d'essais :

Variables	Essai 1	Affichage	Essai 2	Affichage
TauxHoraire	15	300 225	40	600 450
NombreHeure	20		15	
SalaireBrut	300		600	
Impot	75		150	
SalaireNet	225		450	

2.5 Le quiz de révision

Faire la trace des algorithmes suivants

1. Avec le jeu d'essai: 43 25

Algorithme du bouton Somme :

Lis Nombre1, Nombre2
Somme \leftarrow Nombre1 + Nombre 2
Écris Somme

Variables	Essai	Écran

2. Avec le jeu d'essai: 23,99 5

Algorithme du bouton Total :

Lis PrixProduit, Quantite
TotalSansTaxe \leftarrow PrixProduit * Quantite
Taxe \leftarrow TotalSansTaxe * 0,19
TotalAvecTaxe \leftarrow TotalSansTaxe + Taxe
Écris TotalSansTaxe, Taxe, TotalAvecTaxe

Variables	Essai	Écran

3. Modifier l'algorithme du numéro 2 de manière à permettre que le pourcentage de taxe à calculer soit lu aussi au clavier.

Lis PrixProduit, Quantite, Taxe
TotalSansTaxe \leftarrow PrixProduit * Quantite
Taxe \leftarrow TotalSansTaxe * Taxe
TotalAvecTaxe \leftarrow TotalSansTaxe + Taxe
Écris TotalSansTaxe, Taxe, TotalAvecTaxe

4. Lire un solde de compte de banque, un dépôt ainsi qu'un retrait. Après avoir saisi les données, calculer et afficher le nouveau solde du compte.

Entrées	Sorties
SoldeActuel	NouveauSolde
Depot	
Retrait	

Algorithme du bouton Calculer :

Lis SoldeActuel, Depot, Retrait
NouveauSolde \leftarrow SoldeActuel + Depot – Retrait
Écris NouveauSolde

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran
SoldeActuel	100		130	
Depot	50		0	
Retrait	20		40	
NouveauSolde	130		90	

5. Lire trois (3) nombres, calculer le carré de chaque nombre et afficher la somme des carrés.

Entrées	Sorties
Nombre1	Somme
Nombre2	
Nombre3	

Algorithme du bouton Calculer :

Lis Nombre1, Nombre2, Nombre3
Somme \leftarrow Nombre1 * Nombre1 + Nombre2 * Nombre2 + Nombre3 * Nombre3
Écris Somme

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran
Nombre1	10		5	
Nombre2	20		2	
Nombre3	30		8	
Somme	1400		93	

D'autres solutions possibles ?

Lis Nombre1, Nombre2, Nombre3
Carre1 \leftarrow Nombre1 * Nombre1
Carre2 \leftarrow Nombre2 * Nombre2
Carre3 \leftarrow Nombre3 * Nombre3
Somme \leftarrow Carre1 + Carre2 + Carre3
Écris Somme

6. Lire un revenu annuel, un taux d'impôt provincial et un taux d'impôt fédéral. Calculer et afficher le montant total d'impôt à payer et le revenu après impôt.

Entrées	Sorties
RevenuAnnuel	TotalImpot
TauxImpotProv	RevenuNet
TauxImpotFed	

Algorithme du bouton Calculer :

Lis RevenuAnnuel, TauxImpotProv, TauxImpotFed
 $\text{TotalImpot} \leftarrow \text{RevenuAnnuel} * \text{TauxImpotProv} + \text{RevenuAnnuel} * \text{TauxImpotFed}$
 $\text{RevenuNet} \leftarrow \text{RevenuAnnuel} - \text{TotalImpot}$
 Écris RevenuNet, TotalImpot

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran
RevenuAnnuel	1000		1500	
TauxImpotProv	0,15		0,15	
TauxImpotFed	0,20		0,15	
TotalImpot	350		450	
RevenuNet	650		1050	

7. Lire un revenu brut et calculer le montant d'impôt à payer sachant que le taux d'impôt provincial est de 22% et celui du fédéral de 19%. Afficher l'impôt total.

Entrées	Sorties

Algorithme du bouton Calculer :

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran

8. Faire la trace de l'algorithme du bouton Calcul:

Lis Nombre1, Nombre2, Nombre3

Somme \leftarrow Nombre1 + Nombre2 + Nombre3

Moyenne \leftarrow Somme / 3

Produit \leftarrow Nombre2 * Nombre3

Somme \leftarrow Somme + Produit

Écris Somme, Moyenne, Produit

Variables	Essai 1	Écran	Essai 2	Écran
Nombre1	4	45 5 30	9	
Nombre2	5		8	
Nombre3	6		7	
Somme	15 45			
Moyenne	5			
Produit	30			

9. Faire la trace de l'algorithme du bouton Test :

Lis A, B, C

D \leftarrow (A + 2) / (B + 1)

E \leftarrow D + 1

F \leftarrow A + B + D + E

E \leftarrow E + F

Écris D, E, F

Variables	Essai 1	Écran
A	20	
B	10	
C	2	
D		
E		
F		

10. Lire un nombre d'autos et un nombre de bateaux. Calculer et afficher la moyenne d'autos par bateau ainsi que le pourcentage d'occupation sachant que chaque bateau a une capacité de 50 autos.

Entrées	Sorties

Algorithme du bouton Calculer :

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran

11. À partir d'une quantité d'un produit acheté et du prix unitaire de ce produit, calculer le prix de vente du produit en sachant qu'on accorde un escompte de 5% sur la transaction. Afficher le montant de l'escompte et le montant total de la vente.

Entrées	Sorties

Algorithme du bouton Calculer :

Jeu d'essai :

Variables	Essai 1	Écran	Essai 2	Écran

3 Introduction au langage C#

3.1 La conception visuelle avec Microsoft Visual C#.Net

Les programmes (applications) que nous écrirons durant cette session sont des applications Windows, c'est-à-dire un formulaire principal (contrôle *Form*) qui permet d'entrer des données, d'afficher des résultats et munie de boutons de commande pour exécuter les traitements.

Un formulaire contient différents **contrôles**. Un contrôle est un objet visuel qui peut être placé sur un formulaire.

Pour l'instant nous utiliserons 4 contrôles:

Formulaire (*Form*) : la fenêtre principale de l'application

Zone d'édition (*TextBox*) : zone qui permet de saisir ou d'afficher du texte.

Zone de texte (*Label*) : zone d'affichage de texte tel que les titres. Elle est utile pour l'apparence du formulaire et permet d'identifier visuellement les zones d'édition.

Bouton de commande (*Button*) : bouton qui lorsqu'actionné exécute le traitement (les instructions) correspondant.

Pour Visual C#, le formulaire principal lui-même, ainsi que tous les contrôles qu'il contient, sont des **objets**. Un objet est défini à l'aide, en outre, de **propriétés et de méthodes**. Tous les contrôles d'un formulaire sont accessibles par les méthodes du formulaire.

Les **propriétés** permettent de décrire l'allure (ex. couleur, dimension, apparence, etc.) et l'état de l'objet (ex. cliquable ou non, actif ou non, visible ou non).

Les **méthodes** permettent de faire réagir l'objet (ex. couper, copier, coller le contenu), elles correspondent aux traitements définis pour l'objet.

Le formulaire principal ainsi que tous les contrôles de l'application possèdent la propriété **Name** qui permet de l'identifier. Par défaut, lorsqu'on place un contrôle dans le formulaire, Visual C# lui attribue un nom (toujours le type du contrôle suivi d'un incrément: **Form1, Label1, Label2, Label3, TextBox1, TextBox2, Button1, Button2**, etc.).

On doit attribuer des noms significatifs pour chaque contrôle, cela fait partie de la documentation d'un programme. Pour les zones de texte, on peut laisser les noms attribués par **Visual C# (Label1, Label2, etc.)** sauf si on veut les utiliser dans nos instructions; il faut alors les identifier de façon significative.

3.1.1 Convention pour l'attribution du nom des contrôles

Préfixe identifiant le type de contrôle écrit en lettres **minuscules**, suivi d'un ou plusieurs mots significatifs, avec majuscule sur la première lettre de chaque mot. Utiliser des caractères alphabétiques ou numériques sans espaces, sans accent et au singulier. On omet les déterminants (ex. Total des notes s'écrit TotalNote) mais on conserve les prépositions (ex. Les personnes à charge s'écrit PersonneACharge. Les abréviations reconnues sont acceptées (ex. numéro de facture s'écrit NoFacture).

Autres exemples: txtPrenom, txtSalaireBrut, btnCalculer, btnEnregistrer, label1, lblSalaireTotal.

Voici la liste des préfixes les plus utilisés qui spécifient le type du contrôle.

Form	frm	GroupBox	gb
Label	lbl	PictureBox	pb
LinkLabel	ll	Panel	pan
Button	btn	DataGrid	dg
TextBox	txt	ListBox	lb
MainMenu	mnu	CheckListBox	clb
CheckBox	cb	ComboBox	cbo
RadioButton	rb	ListView	lv
		TreeView	tv

Pour modifier la propriété *Name* d'un objet de type contrôle (ou toute autre propriété) :

- **Sélectionner l'objet visuel**
- **Dans le formulaire des propriétés, retrouver la propriété à modifier**
- **Modifier sa valeur**

3.1.2 La table des propriétés

Pour faciliter la conception de notre application avec Visual C#, on construira au début seulement, avec le dessin du formulaire, une table des propriétés c'est-à-dire un tableau regroupant les informations de chaque contrôle de l'application.

Le nom des contrôles visuels d'entrée/sortie inscrits dans cette table pourra être utilisé dans les instructions C# pour référer au contrôle soit en lecture soit en écriture. Ces noms sont utilisés lorsqu'on traduit en C# les instructions **Lis** et **Écris** de notre algorithme.

Le nom des contrôles visuels de commande (bouton) inscrits dans cette table pourra être utilisé dans les instructions C# pour référer au contrôle pour y associer le traitement lors d'un clic.

Table des propriétés

TYPE	PROPRIÉTÉ	NOM
Form	Name	frmCalcul
	Text	Calcul de Salaire
TextBox	Name	txtNom
TextBox	Name	txtTaux
TextBox	Name	txtHeure
TextBox	Name	txtSalBrut
TextBox	Name	txtSalNet
Button	Name	btnCalculer
	Text	Calculer

Exemple :

À partir de l'âge d'une personne, calculer et afficher le nombre de jours, le nombre d'heures, le nombre de minutes et le nombre de secondes qu'elle a vécu. Prenez pour acquis qu'il y a toujours 365 jours dans une année.

Faire l'analyse de problème, la table des propriétés et la conception visuelle.

Entrées	Sorties
Age	NombreJour
	NombreHeure
	NombreMinute
	NombreSeconde

Table des propriétés

TYPE	PROPRIÉTÉ	NOM
Form	Name Text	frmDecomposition Décomposition de l'âge
TextBox	Name	txtAge
TextBox	Name	txtNombreJour
TextBox	Name	txtNombreHeure
TextBox	Name	txtNombreMinute
TextBox	Name	txtNombreSeconde
Button	Name Text	btnCalculer Calculer

Titre

Âge

Calculer

JoursHeures

MinutesSecondes

3.2 Les commentaires et règles d'écriture

De façon à fournir une documentation de base au programme, inscrire, au début du fichier source, un commentaire incluant votre nom, la date et une brève description de l'application.

De plus, avant chacune des méthodes, inscrire le but de la méthode ainsi que l'événement qui la déclenche.

Un commentaire est inclus en C#, entre les signes `/* */` et peuvent s'étendre sur plusieurs lignes.

Sur une seule ligne ou après une instruction C#, on peut utiliser le symbole `//`

3.3 Les variables

Une variable est un nom symbolique pour désigner un espace mémoire réservé pour stocker une valeur. Le type de la variable détermine l'espace mémoire réservé ainsi que l'étendue des valeurs qu'il est possible de stocker dans la variable. Son type est déterminé lors de la déclaration de la variable. La déclaration doit être faite avant la première utilisation de la variable.

Exemple : `int Nombre; // déclaration de la variable entière Nombre`

`Nombre = 5; // utilisation de la variable Nombre`

3.3.1 L'attribution de noms aux variables

Pour utiliser une variable, d'abord lui trouver un nom approprié et significatif. Chaque variable possède un nom. Voir point 2.3.2

Respecter les conventions d'affectation de noms standards recommandées pour C#. Mémoriser les mots clés réservés en C# qui ne peuvent pas être utilisés comme noms de variables.

Règles et recommandations pour l'attribution de noms aux variables :

Règles:

Le nom doit débiter par **une lettre** ou un trait de soulignement;

Le premier caractère doit être suivi par des lettres, des chiffres ou un trait de soulignement;

Les mots clés réservés sont interdits.

Conventions:

Utiliser **une lettre majuscule** pour la **première lettre de chaque mot** formant le nom de la variable;

Éviter les noms de variables commençant par un trait de soulignement, ils sont réservés pour nommer un cas particulier de variable;

Lorsque vous utilisez des abréviations, elles doivent être significatives;

Préfixer de `m_` la variable membre.

3.4 Les types de données

Le tableau suivant répertorie les mots clés des types intégrés C#, qui constituent des alias des types prédéfinis de l'espace de noms **System**.

	Type C#	Alias Type .NET Framework	Plage	Taille
Booléen	bool	System.Boolean	true ou false	
Caractère	char	System.Char	U+0000 à U+ffff	Caractère Unicode 16 bits
	decimal	System.Decimal	$\pm 1.0 \times 10^{-28}$ à $\pm 7.9 \times 10^{28}$ précision 28-29 chiffres	128 bits (16 octets)
Réel	double	System.Double	$\pm 5.0 \times 10^{-324}$ à $\pm 1.7 \times 10^{308}$ précision 15-16 chiffres	64 bits (8 octets)
	float	System.Single	$\pm 1.5 \times 10^{-45}$ à $\pm 3.4 \times 10^{38}$ précision 7 chiffres	32 bits
	byte	System.Byte	0 à 255	Entier 8 bits non signé
	sbyte	System.SByte	-128 à 127	Entier 8 bits signé
	short	System.Int16	-32 768 à 32 767	Entier 16 bits signé
	ushort	System.UInt16	0 à 65 535	Entier 16 bits non signé
Entier	int	System.Int32	-2 147 483 648 à 2 147 483 647	Entier 32 bits signé
	uint	System.UInt32	0 à 4 294 967 295	Entier 32 bits non signé
	long	System.Int64	-9 223 372 036 854 775 808 à 9 223 372 036 854 775 807	Entier 64 bits signé
	ulong	System.UInt64	0 à 18 446 744 073 709 551 615	Entier 64 bits non signé
Chaîne	string	System.String		
	object	System.Object		

Notes

Hormis les types **string** et **object**, tous les types répertoriés dans le tableau sont considérés comme des types simples (ne contenant qu'une seule valeur)

Les mots clés des types C# et leurs alias sont interchangeables. Par exemple, vous pouvez déclarer une variable de type réel à l'aide d'une des deux déclarations suivantes :

```
double x = 123.55;
System.Double x = 123.55;
```

La notation des décimales en C# se fait à l'aide du point (.).

3.4.1 La conversion entre les types numériques :

Il faut porter une attention particulière lorsqu'on utilise des types numériques différents dans une expression.

Un entier n'est pas un réel et un réel n'est pas un entier.

Le type réel a plus de précision que le type entier.

La constante 12 n'a pas la même précision que la constante 12.0.

Il y a conversion automatique ou implicite dans le type ayant le plus de précision si plusieurs types numériques différents sont utilisés dans une expression.

Il y a conversion implicite lors de l'affectation dans une variable du résultat d'une expression si le type du résultat de l'expression est dans un type de moindre précision que celui de la variable réceptrice.

C'est principalement lors d'une division que surgent certains problèmes causés par la conversion implicite.

Par exemple :

Un entier divisé par un entier donne un entier	2 / 5 donne 0	Aucune conversion
Un réel divisé par un réel donne un réel	2.0 / 5.0 donne 0.4	Aucune conversion
Un entier divisé par un réel donne un réel	2 / 5.0 donne 0.4	La valeur 2 converti en réel 2.0
Un réel divisé par un entier donne un réel	2.0 / 5 donne 0.4	La valeur 5 converti en réel 5.0
Un entier affecté à une variable entière	int a = 3;	a vaut 3
Un entier affecté à une variable réelle	double b = 5;	b vaut 5.0
Un réel affecté à une variable entière	int c = 4.5;	c vaut 4
Un réel affecté à une variable réelle	double d = 6.4;	d vaut 6.4

L'utilisation de la conversion explicite permet de préciser un type temporaire qui sera utilisé lors de l'exécution de l'opération.

L'utilisation de la conversion explicite permet de déterminer adéquatement le type d'une variable sans tenir compte de son utilisation dans une expression. Il est inadéquat de modifier le type d'une variable à cause de son utilisation dans une expression.

Prenons l'exemple suivant :

Un local peut contenir au maximum quarante (40) personnes. Nous voulons connaître le pourcentage d'occupation sachant qu'il y a 23 personnes dans ce local.

Le calcul est relativement simple : il suffit de diviser le nombre de personnes actuellement dans le local par le nombre maximum et de multiplier par 100 pour obtenir le pourcentage d'occupation. Ce pourcentage est exprimé sans décimal.

a) En utilisant le type implicite :

23 / 40 * 100
0 * 100
0

b) En utilisant la conversion explicite d'un entier en réel:

(double)23 / 40 * 100	23 / (double)40 * 100	23 / 40 * (double)100
0.575 * 100	0.575 * 100	0 * 100
57.5	57.5	0

L'utilisation de (double) devant une valeur constante ou une variable d'un type numérique de moindre précision permet une conversion explicite qui s'applique à l'opération.

c) En utilisant la conversion explicite d'un réel en entier :

(int)((double)23 / 40 * 100)	(int)(23 / (double)40 * 100)
0.575 * 100	0.575 * 100
57.5	57.5
57	57

L'utilisation de (int) devant une valeur constante ou une variable d'un type numérique de haute précision permet une conversion explicite qui s'applique à l'opération.

d) Une portion d'une méthode en C# illustrant ce principe :

```
int NombreMaximumPersonne = 40;
int NombreActuelPersonne = 23;
int PourcentageOccupation;
```

```
PourcentageOccupation = (int) (NombreActuelPersonne / (double) Nombre MaximumPersonne * 100);
```

```
// puisqu'un nombre de personne sera toujours entier, il serait inadéquat de modifier le type entier en réel
// les trois variables entières occupent 96 octets au lieu de 192 pour des réels .
```

3.5 L'entête de la méthode

Lorsque l'on développe une application pour Windows, le logiciel de développement d'application Microsoft Visual C# écrit pour nous la trame de code nécessaire à nos applications. Il s'occupe entre autre de la programmation de base des formulaires Windows (ex. les icônes de fermeture, de restauration etc.). Cette partie de programmation est lourde et répétitive. Le programmeur n'a plus qu'à inclure le traitement spécifique de son application sous forme de méthode.

Une méthode est un ensemble d'instructions à exécuter suite à un événement ou par appel; l'entête de méthode marque le début de la méthode. Elle indique le qualificateur (public, private, protected), le type de retour de la méthode, le nom de la méthode et entre parenthèse les paramètres qu'elle reçoit lors de son appel.

Exemple :

```
private void btnCalculer_Click(object sender, System.EventArgs e)  
{  
  
}
```

Dans ce segment de programme, la première ligne est la signature de la méthode associée à un bouton nommé btnCalculer qui sera exécutée lorsque l'utilisateur cliquera sur le bouton.

Les instructions comprises dans la méthode sont délimitées par les accolades d'ouverture et de fermeture.

3.6 La déclaration des variables locales à une méthode

Les variables locales à une méthode sont celles que l'on déclare à l'intérieur de la méthode. Elles sont créées à l'appel de la méthode et sont détruites lorsque la méthode se termine. Elles ne sont donc visibles qu'à l'intérieur de la méthode.

Les variables utilisées dans l'algorithme associé à un bouton ont à être déclaré comme variables locales à la méthode.

On déclare les variables locales au début de la méthode en précisant le type, le nom et une valeur initiale si nécessaire.

```
private void btnCalculer_Click(object sender, System.EventArgs e)  
{  
    string Nom;  
    double TauxHoraire, SalaireBrut;  
    double SalaireNet;  
    int NombreHeure;  
    double Impot = 0;           //déclaration et initialisation  
    ....
```

3.7 La traduction de l'instruction algorithmique Lis

Pour qu'une variable locale reçoive la valeur que l'utilisateur aura saisie dans la zone d'édition (TextBox) il faudra **transférer la valeur de la propriété Text du contrôle identifié par son nom (ref Table des propriétés) dans une variable locale** en utilisant la méthode de conversion appropriée au type de la variable locale.

Pour une variable chaîne : aucune conversion nécessaire puisque la propriété Text d'un TextBox est de type chaîne.

Double.TryParse : Convertir la représentation sous forme de chaîne d'un nombre en nombre à virgule flottante double précision équivalent. Sans cette conversion, la variable ne pourra être utilisée dans une expression arithmétique.

Exemple : `Resultat = Double.TryParse(txtTaux.Text, out TauxHoraire);`

Int32.TryParse : Convertir la représentation sous forme de chaîne d'un nombre en entier signé 32 bits équivalent. Sans cette conversion, la variable ne pourra être utilisée dans une expression arithmétique.

Exemple : `Resultat = Int32.TryParse(txtHeure.Text, out NombreHeure);`

Char.TryParse : Convertir la représentation sous forme de chaîne d'un caractère Unicode équivalent.

Exemple : `Resultat = Char.TryParse(txtCode.Text, out Code);`

Toujours vérifier que la conversion a réussi avant de poursuivre le programme.

3.8 La traduction de l'instruction algorithmique d'affectation

Pour traduire en C# l'instruction algorithmique d'affectation, on change simplement le symbole d'affectation (\leftarrow) par le signe d'affectation (assignment) = .

```
// Calcul du salaire brut, de l'impôt et du salaire net
```

```
SalaireBrut = TauxHoraire * Heure;  
Impot = SalaireBrut * 0.25;  
SalaireNet = SalaireBrut - Impot;
```

3.9 La traduction de l'instruction algorithmique Écris

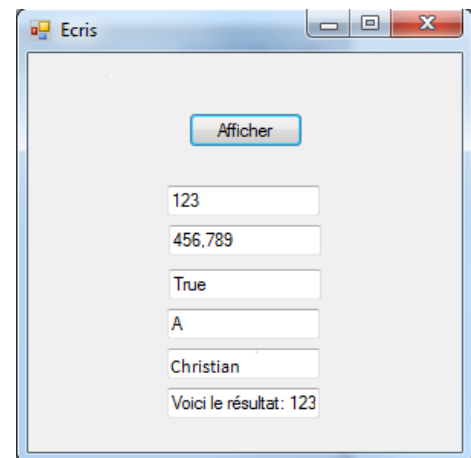
Pour que la méthode affiche/écrive la valeur d'une de nos variables locales dans une zone d'édition (TextBox) il faudra **affecter à la propriété Text du contrôle identifié par son nom (ref Table des propriétés) la valeur de notre variable locale**. On utilise la méthode ToString() de la variable locale à transférer si celle-ci n'est pas de type chaîne.

C'est ici que l'on doit faire la mise en forme des variables à écrire, s'il y a lieu.

Pour l'instant, nous affichons nos variables dans une boîte de texte donc il faut les convertir en chaîne de caractère.

Ex :

```
private void btnAfficher_Click(object sender, EventArgs e)  
{  
    int a = 123;  
    double b = 456.789;  
    bool c = true;  
    char d = 'A';  
    string f = "Christian";  
  
    txtAffA.Text = a.ToString();  
    txtAffB.Text = b.ToString();  
    txtAffC.Text = c.ToString();  
    txtAffD.Text = d.ToString();  
    txtAffF.Text = f;  
    txtAff.Text = "Voici le résultat: " + a.ToString();  
}
```



3.10 Une méthode complète en C#

```
/* Exemple 1 : Traduction des instructions Lis, Écris et d'affectation
Il faut avoir fait la conception visuelle du formulaire et avoir nommé adéquatement chacun des contrôles . */
// Cette méthode calcule le salaire brut et le salaire net selon un taux horaire et un nombre d'heures
// L'impôt est toujours de 25%

private void btnCalculer_Click(object sender, EventArgs e)
{
    // Déclaration des variables locales voir 3.5
    string Nom;
    double TauxHoraire, SalaireBrut;
    double SalaireNet;
    int NombreHeure;
    double Impot;
    bool Resultat; // variable nécessaire pour valider le type des données saisies

    // Lecture du nom; aucune conversion nécessaire voir 3.6
    // la valeur de la propriété Text du contrôle nommé txtNom est affecté à la variable locale Nom

    Nom = txtNom.Text;

    // Lecture du Taux Horaire ; conversion en réel

    Resultat = Double.TryParse(txtTaux.Text, out TauxHoraire); // on tente la conversion d'une
                                                                //chaîne en réel voir 3.6

    if (Resultat == false)                                     // si erreur, on affiche une boîte à message
    {
        MessageBox.Show("Le taux horaire est invalide", // texte du message (à votre choix)
                        "Validation Taux Horaire", // titre de la boîte à message
                        MessageBoxButtons.OK, // un seul bouton OK nécessaire ici
                        MessageBoxIcon.Error); // affiche l'icône d'erreur

        return;
    }

    // Lecture du nombre d'heures et conversion en entier voir 3.6

    Resultat = Int32.TryParse(txtHeure.Text, out NombreHeure);

    if (Resultat == false)
    {
        MessageBox.Show("Le nombre d'heures est invalide",
                        "Validation Heures",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);

        return;
    }

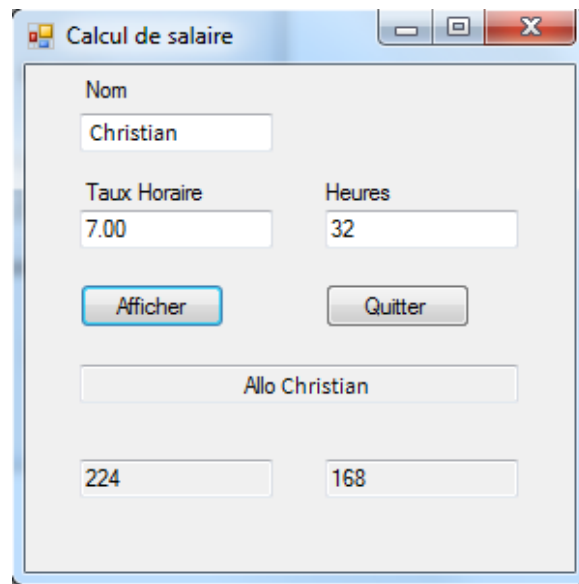
    // Traduction de l'algorithme en C#

    // Calcul du salaire brut, de l'impôt et du salaire net voir 3.7

    SalaireBrut = TauxHoraire * NombreHeure;
    Impot = SalaireBrut * 0.25; //Remarquer l'utilisation du point (.)
    SalaireNet = SalaireBrut - Impot;
    Nom = "Allo " + Nom;
    // Affichage des résultats en transférant la conversion en chaîne (ToString) des variables locales
    // Voir 3.8

    txtSalBrut.Text = SalaireBrut.ToString();
    txtSalNet.Text = SalaireNet.ToString();
    txtNom.Text = Nom;
}
```

Exécution de l'application



The screenshot shows a Windows application window titled "Calcul de salaire". The window has a standard Windows XP-style title bar with minimize, maximize, and close buttons. The main content area is light gray and contains the following elements:

- A label "Nom" above a text input field containing "Christian".
- Two labels, "Taux Horaire" and "Heures", each above a text input field. "Taux Horaire" contains "7.00" and "Heures" contains "32".
- Two buttons: "Afficher" (highlighted with a blue border) and "Quitter".
- A large text area or label displaying "Allo Christian".
- Two text input fields at the bottom, containing the numbers "224" and "168".

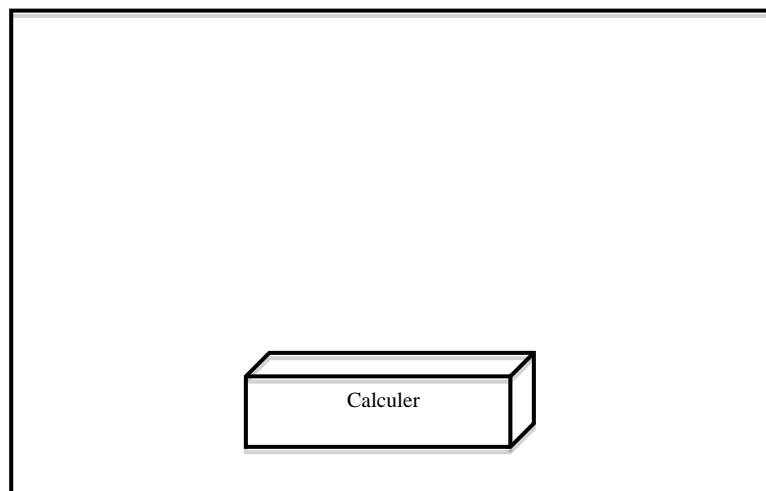
3.11 Le quiz de révision

1- Faire les étapes 1 à 5 pour résoudre le problème suivant.

À partir de la capacité d'un disque rigide en Mo, du nombre de pages d'un livre et du nombre moyen de caractères dans une page, calculer et afficher combien de livres peuvent être entreposés sur ce disque. Après la saisie des informations, on devra appuyer sur le bouton Calcul pour déclencher le traitement.

N.B. : 2 octets = 1 caractère.
 1 KiloOctet (1K) = 1024 octets.
 1 MégaOctet (1 Mo) = 1024 K etc...

Entrées	Sorties



Algorithme du bouton Calculer :

Jeu d'essai :

Variables	Essai 1	Essai 2

Énumérer les variables de votre algorithme et préciser le type de chacune d'elles.

Énumérer les constantes de votre algorithme.

2- Faire le processus de résolution de problème étapes 1 à 5 et l'étape 7

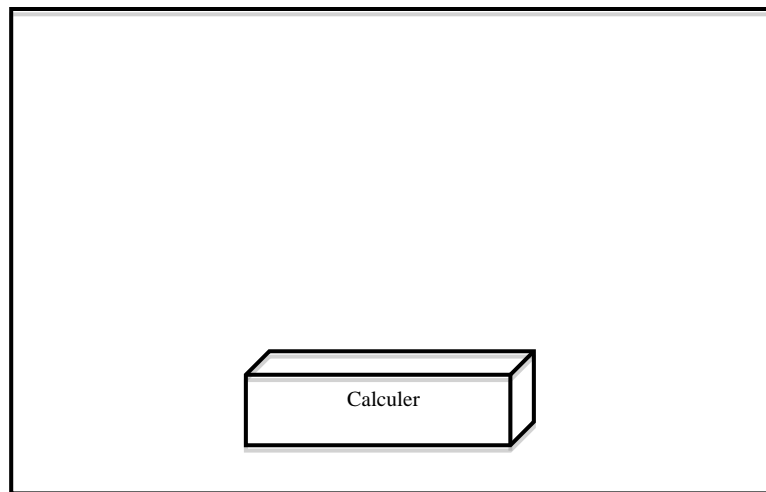
Une compagnie d'assurances automobile souhaite une application pour calculer la prime de ses clients actuels. À partir du nombre d'années d'expérience sans accident, du nombre d'accidents durant la dernière année, la prime d'assurance pour l'année en cours, calculer la nouvelle prime d'après les spécifications suivantes:

- une augmentation de 2% pour tous les clients
- une augmentation de 20,00 \$ par accident
- une diminution de 3,00 \$ par année d'expérience sans accident.

Calculer aussi l'augmentation en % par rapport à l'année en cours ainsi que le montant à payer après avoir ajouté une taxe de 7 % à la nouvelle prime.

Afficher la nouvelle prime, le pourcentage d'augmentation, la taxe et le montant à payer.

Entrées	Sorties



Algorithme du bouton Calculer :

Variables	Essai 1	Essai 2

La traduction en C# de l’algorithme du bouton Calculer :

```
private void btnCalculer_Click(object sender, EventArgs e)
{
```

3.12 Le résumé et le vocabulaire

À la fin de ce chapitre vous devez avoir acquis les notions nécessaires pour développer en C#, à partir d'un énoncé simple, une application Windows. Pour y parvenir, vous appliquez scrupuleusement les huit (8) étapes décrites en 2.1 et vous utilisez le format de présentation obligatoire tel qu'utilisé en 3.10.

Vous avez fait vérifier tous vos exercices et remis les travaux pratiques #1 et #2 tel que demandés.

Vocabulaire :

- | | |
|--|--|
| <ul style="list-style-type: none">• Analyse d'un problème• Entrées et Sorties• Tables de propriétés des contrôles• Nommage des contrôles• Conception visuelle• Algorithme<ul style="list-style-type: none">○ Traitement séquentiel○ Variable entière, réelle, chaîne et caractère○ Nommage des variables○ Constante entière, réelle, chaîne et caractère○ Opérateurs mathématiques○ Priorité des opérateurs mathématiques○ Instruction○ Affectation○ Expression○ Lis○ Écris○ Mots réservés• Trace et jeu d'essais | <ul style="list-style-type: none">• Environnement de développement VisualStudio et C#• Contrôles visuels et leurs propriétés<ul style="list-style-type: none">○ Formulaire (Form)○ Bouton (Button)○ Boîte d'édition (TextBox)○ Étiquette (Label)• Mise au point du visuel de l'application• Déclaration des variables locales• Types des variables : int, double, char et string• Initialisation des variables• Instructions en C#• Utilisation de constantes de type : entier, réel, caractère et chaîne• Méthode• Utilisation de méthodes de conversion des entrées et leur validation• Utilisation de méthodes de conversion des sorties• Utilisation de la méthode statique Show de la classe MessageBox |
|--|--|

4 Structure alternative

Il y a trois (3) types de structures contrôlant l'ordre d'exécution des instructions : séquentielle, alternative et répétitive.

Jusqu'à maintenant, nous avons utilisé les structures séquentielles seulement.

Séquentielle: les instructions sont exécutées les unes à la suite des autres, ligne par ligne.

Alternative: certaines instructions sont exécutées selon une condition et d'autres non.

Répétitive: les mêmes instructions sont répétées un certain nombre de fois selon une condition.

L'ordre d'exécution des instructions ne peut être modifié que par les structures alternatives et les structures répétitives.

Lorsqu'une action dépend d'une **condition** nous devons utiliser une structure alternative.

Une structure alternative est utilisée lorsqu'il est nécessaire de choisir entre **2 ou plusieurs voies possibles de traitement** selon une condition. **Ce ne sont pas toutes les voies qui seront exécutées.** Le but de la structure alternative est de résoudre des problèmes ayant plusieurs choix possibles.

Exemples:

- 1) Tiré de la vie courante:
- | |
|-------------------------------|
| S'il y a une panne électrique |
| Les cours sont suspendus |
| Sinon |
| Les cours ont lieu |

L'instruction **Si** est une instruction conditionnelle qui permet d'exécuter des blocs d'instructions différents selon l'évaluation d'une expression booléenne.

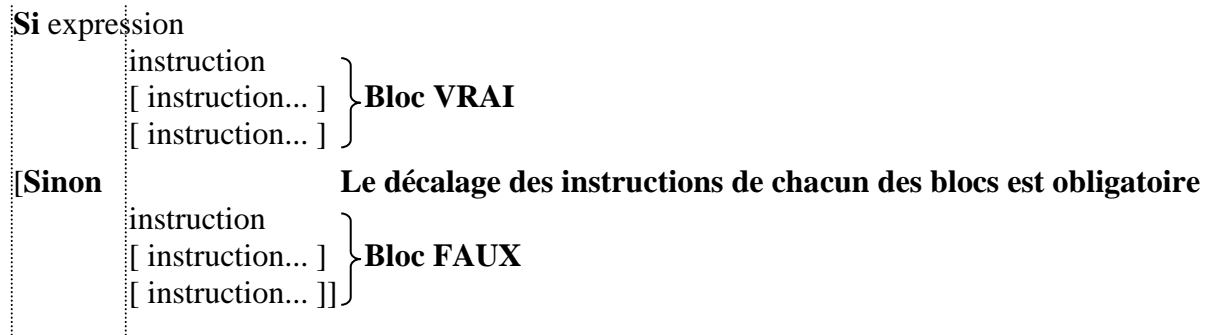
- 2) En informatique: Lire 2 valeurs et afficher la plus petite des deux.

Algorithme du bouton PlusPetit :

Lis	Nombre1, Nombre2
Si	Nombre1 < Nombre2
	PlusPetit ← Nombre1
Sinon	
	PlusPetit ← Nombre2
Écris	PlusPetit

Selon la valeur de la condition (VRAIE ou FAUSSE), on exécute une série d'instructions plutôt qu'une autre. L'utilisation de lignes verticales permet d'aligner adéquatement les décalages et elles sont obligatoires.

4.1 Le format général



- 1) **Expression:** La condition est exprimée à l'aide d'une expression booléenne.
 Le résultat de l'évaluation de l'expression est vrai ou faux.
 On peut relier plusieurs conditions par des ET et/ou des OU.
 Elle peut prendre la forme suivante:

	Variable ou Expression arithmétique	Opérateur relationnel =, <, >, <=, >=, < >, ≠	Variable ou Constante ou Expression arithmétique	Expression
Exemples.				
	a	<	b	a < b
	a + b	>=	b + c	a + b >= b + c
	a	≠	5	a ≠ 5

- 2) **Instruction:**
- ← (Affectation) **Si Tant que Faire Tant que Pour**
- 3) Les blocs VRAI et les blocs FAUX doivent contenir une ou plusieurs instruction(s).
- 4) Le bloc FAUX est optionnel. Selon la logique du problème à résoudre, il y aura ou non un bloc FAUX.
- 5) **Si le résultat de l'évaluation de l'expression est vrai**, les instructions du bloc VRAI s'exécutent normalement jusqu'à la fin de ce bloc; après l'exécution des instructions de ce bloc, le contrôle est transféré à l'instruction qui suit l'instruction **Si**, c'est-à-dire on n'exécute pas le bloc FAUX. **Si le résultat de l'évaluation de l'expression est faux**, le contrôle d'exécution est transféré au bloc FAUX, c'est-à-dire on n'exécute pas le bloc VRAI. Les instructions du bloc FAUX sont exécutées normalement. À la fin de ce bloc, on exécute l'instruction qui suit l'instruction **Si**. S'il y a un bloc FAUX, il y a un bloc VRAI.

4.2 Des exemples

Cas 1: Si expression Instruction	réveille le matin regarde la température Si la température est à la pluie prends le parapluie marche jusqu'au bureau travaille
Cas 2: Si expression Instruction Instruction	réveille le matin regarde la température Si la température est à la pluie prends le parapluie mets les bottes mets l'imperméable marche jusqu'au bureau travaille
Cas 3: Si expression Instruction Sinon Instruction	réveille le matin regarde la température Si la température est à la pluie prends le parapluie Sinon prends les lunettes de soleil marche jusqu'au bureau travaille
Cas 4: Si expression Instruction Instruction Sinon Instruction Instruction	réveille le matin regarde la température Si la température est à la pluie prends le parapluie mets les bottes mets l'imperméable Sinon prends les lunettes de soleil mets les sandales marche jusqu'au bureau travaille

4.3 L'expression simple

L'expression est dite simple si une seule condition est nécessaire pour déterminer les actions à exécuter.

Exemples: **Si** $a > 5$
 Si $a + b = c$

4.4 L'expression complexe

L'expression est dite complexe si plusieurs conditions sont nécessaires pour déterminer les actions à exécuter.

Ce sont des expressions simples reliées par les opérateurs logiques **ET** ou **OU**.

Exemples: **Si** $a > b$ **ET** $a > c$ **OU** $d = 5$
 Si $a + b = c$ **OU** $d = 7$

4.5 L'évaluation des expressions complexes

1. TABLE DE VÉRITÉ: soient la condition A et la condition B

<u>A et B</u>	<u>A ou B</u>
V et V = V	V ou V = V
V et F = F	V ou F = V
F et V = F	F ou V = V
F et F = F	F ou F = F

2. Ordre de priorité

Ordre de priorité	Opérateur
1	()
2	ET
3	OU

4.6 Le quiz de révision

Directive : le résultat des expressions complexes suivantes est-il Vrai ou Faux.

On suppose que $a \leftarrow 5$ $b \leftarrow 6$ $c \leftarrow 8$ $d \leftarrow 7$

- a) **Si** $a > b$ ET $a > c$ OU $d = 5$
- b) **Si** $a + b = c$ OU $d = 7$
- c) **Si** $a \leq 5$ OU $b > 8$ ET $d = 9$ OU $c > 8$
- d) **Si** $(a \leq 5$ OU $b > 8)$ ET $d = 9$ OU $c > 8$

Directive : Faire la trace des algorithmes suivants avec 10 comme valeur lue.

Algorithme du bouton Résultat1	Algorithme du bouton Résultat2	Algorithme du bouton Résultat3
Lis Valeur	Lis Valeur	Lis Valeur
Resultat \leftarrow "Plus grand"	Resultat \leftarrow "Plus grand"	Resultat \leftarrow "Plus grand"
Si Valeur < 15	Chiffre \leftarrow 0	Chiffre \leftarrow 0
Resultat \leftarrow "Plus petit"	Si Valeur < 15	Si Valeur < 15
Ecris Resultat	Resultat \leftarrow "Plus petit"	Resultat \leftarrow "Plus petit"
	Chiffre \leftarrow 15 - Valeur	Chiffre \leftarrow 15 - Valeur
	Ecris Resultat, Chiffre	Ecris Resultat, Chiffre

Directive : Faire le processus de résolution de problème étapes 1 à 5

- a) À partir du prix de base d'un billet et l'âge d'une personne, indiquer le prix final du billet sachant que les gens de l'âge d'or (65 ans et plus) bénéficient d'une réduction de 50% (moitié prix).
- b) Même énoncé que a). De plus, les gens qui ne sont pas de l'âge d'or bénéficient d'un rabais de 1/3. Il faut également afficher le montant du rabais.
- c) Même énoncé que b). De plus, il faut indiquer la catégorie des gens, c'est-à-dire 1 pour l'âge d'or et 2 pour les autres.
- d) Lire un montant et un pourcentage d'escompte, calculer le montant net sachant que l'escompte maximum accordé est 5,25\$. Afficher le montant, l'escompte et le montant net.
- e) À l'entrée, on a un nombre de personnes à charge et un salaire annuel. Tout contribuable a droit à une déduction de 1 000,00\$. Les contribuables qui ont des personnes à charge déduisent 200,00\$ et 80,00\$ pour chaque personne supplémentaire. L'impôt est égal à 17% du revenu imposable.

Afficher le revenu imposable, l'impôt et le salaire net.

Ex.:	Dossier	1234
	Salaire annuel	22 190,00 \$
	Personnes à charge	3
	<i>Le résultat sera:</i>	
	Revenu imposable	20 830,00 \$
	Impôt	3 541,10 \$
	Salaire net	18 648,90 \$

4.7 La traduction de l'instruction Si en C#

L'instruction *if* est une instruction conditionnelle qui permet d'exécuter des blocs d'instructions différents selon l'évaluation d'une expression booléenne.

4.7.1 Les règles d'écriture

L'expression est toujours entre parenthèses.

Par convention, pour faciliter la lecture du code, les instructions sont obligatoirement décalées d'une tabulation. Normalement, le logiciel se charge d'automatiquement décaler les instructions.

Il n'y a pas de point-virgule (;) sur la ligne du *if*.

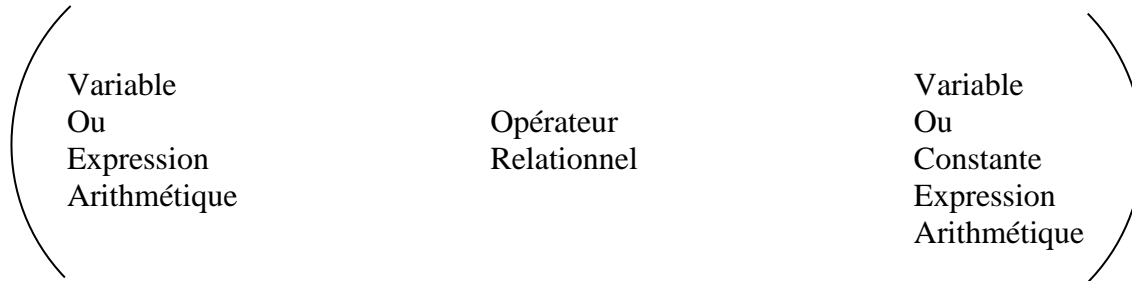
Formats d'écriture :

if (expression) instruction;	Une seule instruction dans le bloc vrai. Pas de bloc faux.
if (expression) { instruction; ... instruction; }	Plusieurs instructions dans le bloc vrai, il faut inclure les instructions entre accolades. Pas de bloc Faux
if (expression) instruction; else instruction;	Une seule instruction dans le bloc vrai et dans le bloc faux.
if (expression) { instruction; ... instruction; } else { instruction; ... instruction; }	Plusieurs instructions dans le bloc vrai et dans le bloc faux, chaque bloc est délimité par des accolades.
On ne doit jamais écrire :	<pre>if(expression) { // avec aucune instruction } else instruction;</pre>

4.7.2 L'expression

L'expression est toujours entre ()

Format général d'une expression :



4.7.3 Les opérateurs relationnels

Les opérateurs relationnels sont:

<=	plus petit ou égal
>=	plus grand ou égal
<	plus petit
>	plus grand
!=	différent (n'est pas égal)
==	égal (deux fois le symbole = sans espace)

Attention: Distinguer l'opérateur d'affectation de l'opérateur relationnel

Exemples: (Age >= 65)
 (Prix > 5.25)
 (Nombre == 25) // Attention à l'opérateur relationnel d'égalité
 (A != B)

4.7.4 Les opérateurs logiques

Les opérateurs logiques sont:

&&	(et)
	(ou)

Ils sont utilisés pour relier deux ou plusieurs conditions dans une expression complexe.

if (Age < 10 || Age > 20)

if (Age >= 12 && Age <= 18)

4.8 La validation des entrées

Nous avons déjà appliqué une certaine forme de validation à la lecture en transférant la donnée saisie dans la propriété Text du TextBox dans la variable locale de la méthode qui traite cette donnée avec la méthode de conversion appropriée. Cette validation ne fait que vérifier que la donnée saisie corresponde bien au type de donnée attendue. Cette validation n'est pas exprimée dans l'algorithme car c'est une caractéristique du langage C#. En C#, le seul type pouvant être lu ou écrit sans conversion, dans une boîte d'édition est la chaîne de caractères. Il faut donc en C#, convertir, ce qui oblige la vérification du résultat de la conversion. C'est un problème propre au langage.

La validation des entrées est **l'action de s'assurer que les valeurs saisies par l'utilisateur à l'exécution d'une application correspondent bien aux valeurs attendues par le programme et non pas au type**. Si la valeur saisie par l'utilisateur ne correspond pas à une valeur attendue, le programme doit en aviser l'utilisateur en affichant un message explicite et rejeter la transaction, donc arrêter l'exécution du traitement.

La validation des entrées est exprimée dans l'algorithme.

Par exemple :

Algorithme du bouton Vote :

Sans validation des entrées

```
Lis Age
Si Age >= 18
    DroitVote ← Vrai
Sinon
    DroitVote ← Faux
```

Algorithme du bouton Vote :

Avec validation des entrées

```
Lis Age
Si Age < 0 OU Age > 120
    Erreur
    Terminer
Si Age >= 18
    DroitVote ← Vrai
Sinon
    DroitVote ← Faux
```

Exemple:

À partir d'un numéro de siège, du prix de base d'un billet, d'un code de taxe (T=taxable, N=non taxable) et du nom du client, on veut afficher un billet de spectacle qui contiendra le nom du client, le numéro de siège et le prix réel du billet, la taxe et le prix à payer. Les numéros de siège possibles sont de 1 à 50. Le prix de base ne peut être inférieur à 25,00\$ ni excéder 200,00 \$. Le prix du billet augmente de 10% pour les sièges de 1 à 25. Le taux de taxe est de 7 % pour les spectacles taxables.

Analyse:	Entrées	Sorties
	Nom	Nom
	NumeroSiege	NumeroSiege
	PrixBase	PrixReel
	CodeTaxe	Taxe
		MontantAPayer

Algorithme:

```

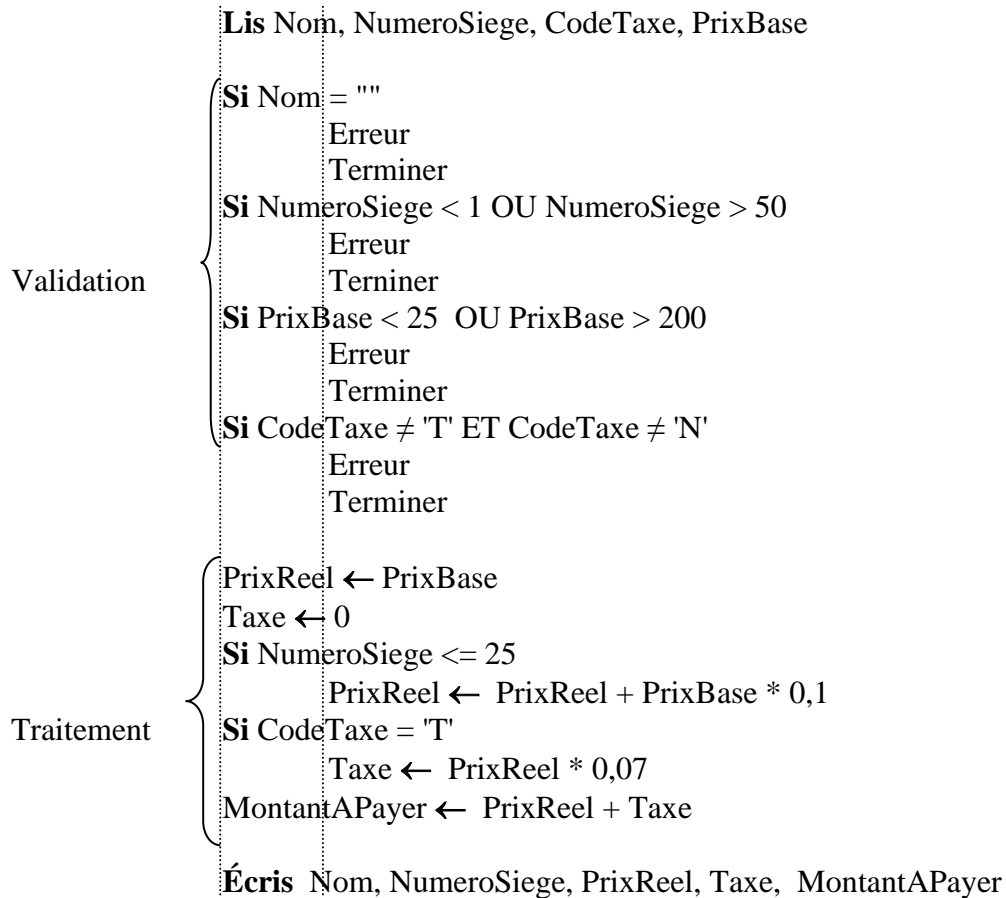
Lis Nom, NumeroSiege, CodeTaxe, PrixBase

Validation {
  Si Nom = ""
    Erreur
    Terminer
  Si NumeroSiege < 1 OU NumeroSiege > 50
    Erreur
    Terminer
  Si PrixBase < 25 OU PrixBase > 200
    Erreur
    Terminer
  Si CodeTaxe ≠ 'T' ET CodeTaxe ≠ 'N'
    Erreur
    Terminer
}

Traitement {
  Si NumeroSiege ≤ 25
    PrixReel ← PrixBase + PrixBase * 0,1
  sinon
    PrixReel ← PrixBase
  Si CodeTaxe = 'T'
    Taxe ← PrixReel * 0,07
  sinon
    Taxe ← 0
  MontantAPayer ← PrixReel + Taxe
  Écris Nom, NumeroSiege, PrixReel, Taxe, MontantAPayer
}

```

Il peut exister plusieurs algorithmes pour résoudre un même problème.



Dans la partie traitement de cet algorithme, il n'y a pas de bloc faux. Des affectations préalables ont été effectuées, ce qui, dans ce cas-ci remplace les blocs faux de la solution précédente.

ATTENTION AUX DÉCALAGES OBLIGATOIRES.

```

private void btnEmission_Click(object sender, EventArgs e)
{
    // *****Déclaration des variables locales*****

    string Nom;
    char CodeTaxe;
    int NumeroSiege;
    double PrixBase, PrixReel, Taxe, MontantAPayer;
    bool Resultat;

    // ***** Lecture et validation*****

    // Lecture du Nom

    Nom = txtNom.Text;

    // Lecture du Numéro de siège

    Resultat = Int32.TryParse(txtSiege.Text, out NumeroSiege);
    if(Resultat == false)
    {
        MessageBox.Show("Le numéro de siège est invalide",
                        "Validation du Numéro de siège",
                        MessageBoxButtons.OK , MessageBoxIcon.Stop);

        txtSiege.Focus();
        return;
    }

    // Lecture du Prix de Base

    Resultat = Double.TryParse(txtPrixBase.Text, out PrixBase);

    if(Resultat == false)
    {
        MessageBox.Show("Le prix de base est invalide" ,
                        "Validation du Prix de Base",
                        MessageBoxButtons.OK , MessageBoxIcon.Stop);

        txtPrixBase.Focus();
        return;
    }

    // Lecture du Code de taxe

    Resultat = Char.TryParse(txtCodeTaxe.Text, out CodeTaxe);
    if(Resultat == false)
    {
        MessageBox.Show("Le code de taxe est invalide" ,
                        "Validation du Code Taxe",
                        MessageBoxButtons.OK , MessageBoxIcon.Stop);

        txtCodeTaxe.Focus();
        return;
    }
}

```



```

// Les validations
if (Nom == "")
{
    MessageBox.Show("Le nom est obligatoire", "Validation du Nom",
                    MessageBoxButtons.OK, MessageBoxIcon.Stop);
    txtNom.Focus();
    return;
}
if (NumeroSiege < 1 || NumeroSiege > 50)
{
    MessageBox.Show("Le numéro du siège doit être entre 1 et 50",
                    "Validation du Numéro de siège",
                    MessageBoxButtons.OK, MessageBoxIcon.Stop);
    txtSiege.Focus();
    return;
}
if (PrixBase < 25 || PrixBase > 200)
{
    MessageBox.Show("Le prix de base doit être entre 25 et 200 $",
                    "Validation du Prix de base",
                    MessageBoxButtons.OK, MessageBoxIcon.Stop);
    txtPrixBase.Focus();
    return;
}
// Transformation du code de taxe en Majuscule
CodeTaxe = char.ToUpper(CodeTaxe);
if (CodeTaxe != 'T' && CodeTaxe != 'N')
{
    MessageBox.Show("Le code doit être N ou T",
                    "Validation du Code Taxe",
                    MessageBoxButtons.OK, MessageBoxIcon.Stop);
    txtCodeTaxe.Focus();
    return;
}

```

```

// ***** Traitement *****

if (NumeroSiege <= 25)
    PrixReel = PrixBase + PrixBase * 0.1;
else
    PrixReel = PrixBase;

// pour les montants monétaires il faut arrondir

PrixReel = Math.Round(PrixReel,2);

if (CodeTaxe == 'T')
{
    Taxe = PrixReel * .07;
    Taxe = Math.Round(Taxe,2);
}
else
    Taxe = 0;
MontantAPayer = PrixReel + Taxe;

// ***** Affichage des résultats*****

txtNomBillet.Text = Nom;
txtNumeroSiegeBillet.Text = NumeroSiege.ToString();
txtPrixReel.Text = PrixReel.ToString();
txtTaxe.Text = Taxe.ToString()
txtMontantAPayer.Text = MontantAPayer.ToString();
}

```

Exemple d'exécution du programme :

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À:

Prix: Taxe:

Validation du Nom

Le nom est obligatoire

OK

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À:

Prix: Taxe:

Validation du Code Taxe

Le code doit être N ou T

OK

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À:

Prix: Taxe:

Validation du Prix de base

Le prix de base doit être entre 25 et 200 \$

OK

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À: Numéro du siège:

Prix: Taxe: Montant à payer:

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À: Numéro du siège:

Prix: Taxe: Montant à payer:

Exemple de validation

Nom du Client:

Prix de base du billet: Siège: Taxable:

Émission du billet

Billet de spectacle

À: Numéro du siège:

Prix: Taxe: Montant à payer:

4.9 La formule de l'arrondi

Nous désirons arrondir la valeur réelle suivante : 1234,567 à deux chiffres après la virgule soit : 1234,57

Il faut passer par quatre (4) étapes :

- | | | |
|--|---|----------|
| 1. Déplacer la virgule de deux (2) chiffres vers la droite en multipliant par 100 | → | 123456,7 |
| 2. Additionner la valeur 0,5 | → | 123457,2 |
| 3. Passer du type réel au type entier pour perdre tous les chiffres après la virgule | → | 123457 |
| 4. Revenir au type réel (123457,00) et diviser par 100 | → | 1234,57 |

$$(\text{int})((\text{ValeurAArrondir} * 100) + 0.5)/100.0$$

**N.B. pour arrondir à un chiffre après la virgule : on multiplie et divise par 10;
pour arrondir à trois (3) chiffres après la virgule : on utilise 1000.**

Lors de la traduction, on met toujours au bon endroit la formule de l'arrondi. On peut utiliser une méthode d'arrondi si cette méthode est disponible dans le langage de programmation utilisé et qu'elle satisfait aux exigences de l'entreprise.

4.10 Le quiz de révision

Ce quiz permet de se préparer adéquatement à l'examen #1

1- Pour chacun des énoncés suivants, indiquer le type de variable approprié pour représenter :
(Note : Les réponses possibles sont: réel, entier, caractère ou chaîne)

- a) La lettre la plus souvent utilisée dans un texte en français. _____
- b) Le nombre d'humains sur Terre estimé à 7 milliards au 31 octobre 2011. _____
- c) L'âge de ton meilleur ami. _____
- d) Le numéro du cours de Programmation I : 420-3A6-LL. _____

2- Répondre aux questions suivantes :

a) Quelle est l'extension du fichier qui représente une solution C# et qui permet d'ouvrir celle-ci à l'aide de Visual Studio?

b) Qu'indique l'extension de fichier .exe ?

c) Que contient le fichier frmTP2.cs de votre TP2?

3- Compléter les phrases suivantes permettant de décrire la convention de nommage des contrôles exigée lors de la conception visuelle.

Le nom d'un contrôle (ex. zone d'édition de texte) doit être _____ et précédé par un _____. Chacune des premières lettres de chaque _____ composant le nom du contrôle doit être en majuscules. Il ne doit y avoir aucun _____ et aucune _____.

4- Le code source suivant génère une erreur à la compilation. Trouver cette erreur et corriger le code source pour permettre la compilation. L'erreur est sur une des lignes numérotées.

	private void btnCalculer_Click(object sender, EventArgs e)
1	{
2	int a=2, b=3, c ;
3	double d = 12,34;
4	
5	c = a*a + b*b;
	}

Voici le message d'erreur reçu :

Seuls une assignation, un appel, un incrément, un décrétement et des expressions d'objet new peuvent être utilisés comme instruction.

Piste de recherche : Identifier et vérifier chacune des instructions.

5- Soit l'algorithme suivant:

```
A ← 6
B ← 4
C ← 12
D ← 3
E ← 2
D ← A + (C - 3) / D
E ← 40 - A * A
C ← B + E2
A ← 21 % C
E CRIS A, B, C, D, E
```

Faire la trace de cet algorithme.

6- Quel est l'effet de l'instruction?

Left = Left - 10;

- a) Déplacer la fenêtre de 10 pixels vers la gauche
- b) Déplacer la fenêtre de 10 pixels vers la droite
- c) Déplacer un bouton de 10 pixels vers la gauche
- d) Déplacer un bouton de 10 pixels vers la droite

Réponse : _____

Quel est l'effet de l'instruction?

btnCalculer.Top = btnCalculer.Top + 15;

- a) Déplacer la fenêtre de 15 pixels vers le haut
- b) Déplacer la fenêtre de 15 pixels le bas
- c) Déplacer le bouton btnCalculer de 15 pixels vers le haut
- d) Déplacer le bouton btnCalculer de 15 pixels vers le bas

Réponse : _____

Quel est l'effet de l'instruction?

Width = Width + 10;

- a) Augmenter la largeur de la fenêtre de 10 pixels.
- b) Augmenter la hauteur de la fenêtre de 10 pixels.
- c) Diminuer la hauteur de la fenêtre de 10 pixels.
- d) Augmenter la hauteur d'un bouton de 10 pixels.

Réponse : _____

7- Énoncé du problème : Calculatrice à bois franc

Un installateur de plancher de bois franc vous demande d'écrire un programme pour calculer et afficher le nombre de paquets de bois nécessaires, le coût total du bois et le coût de la main-d'œuvre pour recouvrir une salle rectangulaire à partir des informations suivantes:

- La largeur de la salle à recouvrir (en mètres)
- La longueur de la salle à recouvrir (en mètres)
- La quantité de bois dans un paquet (en mètres carrés)
- Le prix d'un paquet de bois
- Le salaire de l'installateur à l'heure (Taux horaire)
- Le nombre de paquets installés à l'heure

Note:

1. L'installateur a besoin de 5% de plus de bois pour compenser les différentes pertes lors des coupes.
2. Il est possible d'acheter une fraction d'une boîte de bois. (Le marchand autorise l'ouverture des boîtes)

Compléter l'analyse et écrire l'algorithme.

Analyse

Entrées		Sorties	

Conception visuelle :

Algorithme du bouton Calculer

5 Compteur, accumulateur et variable membre

5.1 Les compteurs et les accumulateurs

5.1.1 Les accumulateurs

Un accumulateur est une variable dans laquelle on accumule des valeurs, par exemple, un compte en banque. À chaque dépôt, la banque accumule la valeur de notre dépôt.

Partons du problème suivant :

À partir du prix de base d'un billet et l'âge d'une personne, indiquer le prix final du billet sachant que les gens de l'âge d'or (65 ans et plus) bénéficient d'une réduction de 50% (moitié prix). De plus, afficher le montant total de tous les billets émis.

Pour résoudre la dernière phrase, on utilise un accumulateur dans lequel le prix final de chaque billet est additionné. Cet accumulateur peut servir dans des expressions arithmétiques. On détermine, s'il y a lieu, à quel moment le résultat en sortie sera affiché, le plus souvent à l'aide de bouton donc d'un formulaire avec plusieurs boutons de traitement différents

5.1.2 Les compteurs

Un compteur est une variable dans laquelle on compte un nombre d'éléments ou d'événements donnés, par exemple, le nombre total de billets émis.

Partons du problème suivant :

À partir du prix de base d'un billet et l'âge d'une personne, indiquer le prix final du billet sachant que les gens de l'âge d'or (65 ans et plus) bénéficient d'une réduction de 50% (moitié prix). De plus, afficher le nombre total de billets émis.

Pour résoudre la dernière phrase, on utilise un compteur que l'on incrémente à chaque billet traité. Ce compteur peut servir dans des expressions arithmétiques. On détermine, s'il y a lieu, à quel moment le résultat en sortie sera affiché, le plus souvent à l'aide de bouton donc d'un formulaire avec plusieurs boutons de traitement différents.

5.2 La variable locale

La variable locale est une variable déclarée et utilisée à l'intérieur d'une méthode. Sa durée de vie est très courte et volatile : allouée dans la méthode et détruite lorsque la méthode se termine. Elle n'est connue que dans la méthode qui la déclare. Une variable déclarée dans une méthode **n'est pas visible ni utilisable dans une autre méthode**.

La **variable locale** est allouée en mémoire à chaque fois que la méthode, dans laquelle elle est déclarée, est ré-exécutée.

Si la première utilisation d'une variable locale n'est pas une affectation, **il est nécessaire de l'initialiser**.

En effet, la déclaration d'une variable réserve l'espace nécessaire en mémoire. La valeur de la variable est indéterminée.

L'initialisation d'une variable place en mémoire une valeur spécifique avant sa première utilisation.

```
Exemple :      double VarA;           // Déclaration
                int VarB=0           // Déclaration et initialisation

                VarA = VarA + 1;      // invalide : VarA n'est pas initialisée

                VarA = 125;           // initialisation par une affectation
```

5.3 La variable membre d'une classe

Dans la phrase : la voiture est un moyen de locomotion.

L'emploi du mot voiture fait référence à des caractéristiques et des comportements communs dont l'ensemble est différent de celui, par exemple, d'un bateau. Imaginez que l'on ne puisse pas classer ces caractéristiques et ces comportements en concepts nommés: au lieu de dire voiture, il faudrait énumérer toutes les significations du mot voiture.

Toutes les voitures ont des comportements communs : elles démarrent, s'arrêtent, clignotent, ...

Toutes les voitures ont des caractéristiques communes : numéro de série, couleurs, options, propriétaire, ...

Vous employez le mot voiture pour désigner une voiture en général.

En **programmation orienté objet**, la description des comportements et des caractéristiques d'une voiture s'appelle : la **classe** voiture.

Dans la phrase : ma Lancer est rouge.

L'emploi du nom Lancer fait référence à une voiture en particulier qui possède ses propres valeurs pour chaque caractéristique la décrivant.

En **programmation orienté objet**, une voiture possédant ses propres caractéristiques s'appelle : un **objet** voiture.

Une **classe** est une bulle de programmation, un concept nommé qui décrit un ensemble de comportements communs s'appliquant sur des caractéristiques dont les valeurs sont propres à chaque **objet**.

Une variable déclarée à l'intérieur d'une classe (et non à l'intérieur d'une méthode) s'appelle une **variable membre** : c'est une variable qui possède une valeur qui caractérise l'objet. Elle est créée lorsqu'on instancie un **objet** particulier de cette **classe**. Sa durée de vie et sa visibilité sont associées à l'**objet**. Elle est visible et **utilisable par toutes les méthodes membres de la classe**.

La **variable membre** est allouée en mémoire lorsque l'**objet** est instanciée, c'est-à-dire lorsque l'espace mémoire est réservée pour l'objet.

Lorsque vous créez une application Windows en Visual C#, vous définissez une nouvelle classe Form ayant, en plus des caractéristiques de base pour tous les formulaires, les caractéristiques propres à votre formulaire (frmXXXX) qui regroupe variables membres et les méthodes membres de cette nouvelle classe.

Lorsque vous démarrez votre application, un objet de votre propre classe est instancié par le code contenu dans le fichier **Program.cs** rendant ainsi disponible les variables membres et les méthodes membres.

Nous pouvons utiliser les variables membres, en autre, pour partager des valeurs entre deux ou plusieurs méthodes membres d'une même classe. Nous utiliserons des variables membres pour partager des valeurs entre deux ou plusieurs méthodes associées à des boutons.

Il est obligatoire de faire débiter le nom des variables membres par le préfixe m_.

5.4 Un exemple nécessitant des compteurs et des accumulateurs

À partir d'un numéro de siège, du prix de base d'un billet, d'un code de taxe (T=taxable, N=non taxable) et du nom du client, on veut afficher un billet de spectacle qui contiendra le nom du client, le numéro de siège et le prix réel du billet, la taxe et le prix à payer. Les numéros de siège possibles sont de 1 à 50. le prix de base ne peut être inférieur à 25,00\$ ni excéder 200,00 \$. Le prix du billet augmente de 10% pour les sièges de 1 à 25. Le taux de taxe est de 7 % pour les spectacles taxables.

De plus, on veut connaître à la fin de la journée seulement, le nombre de billets vendus, le nombre de billets taxés, le nombre de billets non taxés, le montant total de taxe perçue et le montant total d'argent en caisse.

5.4.1 L'analyse:

Entrées	Sorties	
Nom	NomBillet	} Par Billet
NumeroSiege	NumeroSiegeBillet	
PrixBase	PrixReel	
CodeTaxe	Taxe	
	MontantAPayer	
	NombreTotalBillet	} À la fermeture
	NombreBilletTaxe	
	NombreBilletNonTaxe	
	TotalTaxePercu	
	TotalArgent	

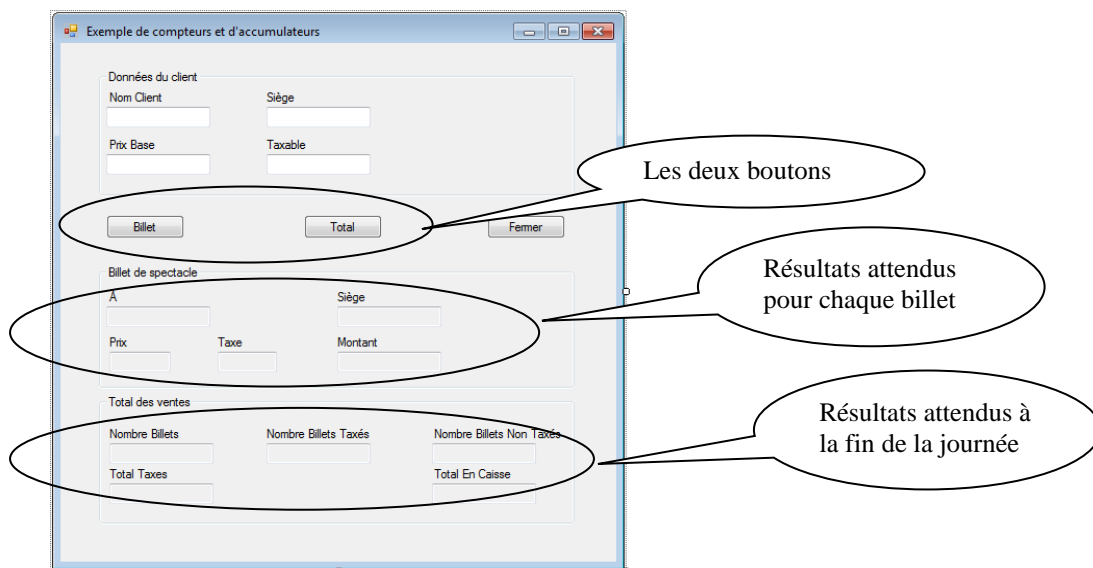
Remarquer qu'il y a deux types de sortie : par billet (au fur et à mesure d'une vente) et à la fermeture (à la fin de la journée,

Lorsqu'on accumule des valeurs, le résultat de cette accumulation peut être utilisé au fur et à mesure ou bien seulement à la fin d'une période donnée (à la fin de la journée par exemple).

Il est très important de bien faire cette distinction lors de l'analyse du problème.

5.4.2 La conception visuelle de l'application

Le résultat de l'analyse nous permet de comprendre qu'il y a deux traitements différents : un par billet et un à la fin de la journée. Pour exécuter ces deux méthodes, nous aurons besoin de deux boutons différents.



5.4.3 Les variables membres nécessaires

Lors de l'analyse du problème, nous avons identifié deux types de données en sortie : celles concernant un seul billet et celles concernant le bilan de fin de journée.

Pour les données en sortie concernant un seul billet nous utiliserons des variables locales. Ces données n'ont pas à être partagées entre plusieurs méthodes.

Pour les données en sortie concernant le bilan de fin de journées nous utiliserons des variables membres puisque ces données seront partagées entre plusieurs méthodes. La méthode associée au bouton Émission utilisera les variables membres pour accumuler et compter les données. La méthode associée au bouton Total utilisera ces mêmes variables membres pour afficher les résultats de fin de journée.

Il est à noter que toutes les variables membres ne sont pas nécessairement des sorties identifiées lors de l'analyse. Ces variables membres peuvent être nécessaires seulement pour arriver au résultat attendu, par exemple lors du calcul d'un pourcentage. Nous y reviendrons plus loin.

Voici les variables membres nécessaires :

Un compteur du nombre de billets : m_NombreTotalBillet

Un compteur du nombre de billets taxables : m_NombreBilletTaxe

Un compteur du nombre de billets non taxables : m_NombreBilletNonTaxe

Un cumulateur des taxes perçues : m_TotalTaxePercue

Un cumulateur de l'argent en caisse : m_TotalArgent

Puisque notre solution utilise deux boutons, nous aurons deux algorithmes à écrire. De plus, dans notre solution nous utilisons des variables membres, il nous faudra donc un troisième algorithme pour initialiser ces variables membres.

5.4.4 L'algorithme Constructeur

```
m_NombreTotalBillet ← 0  
m_NombreBilletTaxe ← 0  
m_NombreBilletNonTaxe ← 0  
m_TotalTaxePercue ← 0  
m_TotalArgent ← 0
```

Le constructeur d'une classe est la méthode membre qui est toujours exécutée lorsque l'objet est instancié.

Cette méthode porte toujours le nom de la classe.

Cette méthode est déjà présente dans la définition de notre classe qui décrit notre formulaire. Nous pouvons y ajouter ce que nous voulons qui soit automatiquement fait lorsqu'on exécute notre projet. Cela nous évite d'avoir à insérer un bouton juste pour initialiser notre formulaire.

5.4.5 L'algorithme du bouton Emission

```
    Lis Nom, NumeroSiege, CodeTaxe, PrixBase  
Validation {  
    Si Nom = ""  
        Erreur  
        Terminer  
    Si NumeroSiege < 1 OU NumeroSiege > 50  
        Erreur  
        Terminer  
    Si PrixBase < 25 OU PrixBase > 200  
        Erreur  
        Terminer  
    Si CodeTaxe <> 'T' ET CodeTaxe <> 'N'  
        Erreur  
        Terminer  
    Si NumeroSiege <= 25  
        PrixReel = PrixBase + PrixBase * 0.1;  
    sinon  
        PrixReel = PrixBase;  
    Si CodeTaxe = 'T'  
        Taxe = PrixReel * .07;  
        m_NombreBilletTaxe = m_NombreBilletTaxe + 1  
        m_TotalTaxePercue = m_TotalTaxePercue + Taxe  
    sinon  
        Taxe = 0;  
        m_NombreBilletNonTaxe = m_NombreBilletNonTaxe + 1;  
    PrixTotal = PrixReel + Taxe;  
    m_TotalArgent = m_TotalArgent + PrixTotal  
    m_NombreTotalBillet = m_NombreTotalBillet + 1  
    Ecris NomBillet, NumeroSiegeBillet, PrixReel, Taxe, PrixTotal  
Traitement }
```

Ne pas oublier les décalages pour les blocs vrais et les blocs faux.

5.4.6 L'algorithme du bouton Total des ventes

Écris **m_NombreTotalBillet** , **m_NombreBilletTaxe** , **m_NombreBilletNonTaxe** ,
m_TotalTaxePercue , **m_TotalArgent**

Noter bien l'utilisation des mêmes variables membres dans les deux algorithmes.

5.4.7 La traduction en C#

//La déclaration des variables membres dans le fichier frmXXX.cs

```
namespace Exemple3
{
    public partial class frmExemple3 : Form
    {

        // déclaration des variables membres
        // elles ne peuvent pas être initialisées ici

        int m_NombreTotalBillet;
        int m_NombreBilletTaxe;
        int m_NombreBilletNonTaxe;
        double m_TotalTaxePercu;
        double m_TotalArgent;
```

//L'initialisation des variables membres dans la méthode constructeur de la classe (frmXXX.cs)

```
public frmExemple3()
{
    InitializeComponent();

    // on insère notre code ici
    // initialisation des variables membres
    m_NombreTotalBillet = 0;
    m_NombreBilletTaxe = 0;
    m_NombreBilletNonTaxe = 0;
    m_TotalTaxePercu = 0;
    m_TotalArgent = 0;
}
```

//La méthode du bouton Émission (frmXXX.cs)

```
private void btnEmission_Click(object sender, EventArgs e)
{
    // Déclaration des variables locales

    string Nom;
    char CodeTaxe;
    int NumeroSiege;
    double PrixBase, PrixReel, Taxe, PrixTotal;
    bool Resultat;

    // ***** Lectures *****

    Nom = txtNom.Text;

    Resultat = Int32.TryParse(txtSiege.Text, out NumeroSiege);
    if(Resultat == false)
    {
        MessageBox.Show("Le numéro de siège est invalide",
            "Validation du Numéro de siège",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
    }
}
```



```

        txtSiege.Focus();
        return;
    }

    Resultat = Double.TryParse(txtPrixBase.Text, out PrixBase);
    if(Resultat == False)
    {
        MessageBox.Show("Le prix de base est invalide",
            "Validation du Prix de Base",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtPrixBase.Focus();
        return;
    }
    Resultat = Char.TryParse(txtCodeTaxe.Text, out CodeTaxe);
    if(Resultat == false)
    {
        MessageBox.Show("Le code de taxe est invalide",
            "Validation du Code Taxe",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtCodeTaxe.Focus();
        return;
    }

    // ***** Les validations *****
    CodeTaxe = char.ToUpper(CodeTaxe);
    if (CodeTaxe != 'T' && CodeTaxe != 'N')
    {
        MessageBox.Show("Le code doit être N ou T",
            "Validation du Code Taxe",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtCodeTaxe.Focus();
        return;
    }
    if (Nom == "")
    {
        MessageBox.Show("Le nom est obligatoire",
            "Validation du Nom",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtNom.Focus(); // permet de positionner le curseur
        return;
    }
    if (NumeroSiege < 1 || NumeroSiege > 50)
    {
        MessageBox.Show("Le numéro du siège doit être entre 1 et 50",
            "Validation du Numéro de siège",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtSiege.Focus();
        return;
    }
    if (PrixBase < 25 || PrixBase > 200)
    {
        MessageBox.Show("Le prix de base doit être entre 25 et 200 $",
            "Validation du Prix de base",
            MessageBoxButtons.OK,
            MessageBoxIcon.Stop);
        txtPrixBase.Focus();
        return;
    }

    // ***** Traitement *****

```

```

        if (NumeroSiege <= 25)
            PrixReel = PrixBase + PrixBase * 0.1m;
        else
            PrixReel = PrixBase;
        PrixReel = Math.Round(PrixReel, 2);
        if (CodeTaxe == 'T')
        {
            Taxe = PrixReel * .07m;
            Taxe = Math.Round(Taxe, 2);
            m_NombreBilletTaxe = m_NombreBilletTaxe + 1;
            m_TotalTaxePercu = m_TotalTaxePercu + Taxe;
        }
        else
        {
            Taxe = 0;
            m_NombreBilletNonTaxe = m_NombreBilletNonTaxe + 1;
        }
        PrixTotal = PrixReel + Taxe;
        m_TotalArgent = m_TotalArgent + PrixTotal;
        m_NombreTotalBillet = m_NombreTotalBillet + 1;

        // Affichage des résultats

        txtNomBillet.Text = Nom;
        txtNumeroSiegeBillet.Text = NumeroSiege.ToString();
        txtPrixReel.Text = PrixReel.ToString();
        txtTaxe.Text = Taxe.ToString();
        txtMontantAPayer.Text = PrixTotal.ToString();

        btnTotaux.Visible=true;
    }

```

//La méthode du bouton Quitter (frmXXX.cs)

```

private void btnQuitter_Click(object sender, EventArgs e)
{
    Application.Exit();
}

```

//La méthode du bouton Totaux (frmXXX.cs)

```

private void btnTotaux_Click(object sender, EventArgs e)
{
    gbEntree.Visible = false;
    btnEmission.Visible = false;
    btnTotaux.Visible = false;
    gbBillet.Visible = false;
    gbTotal.Visible = true;
    txtNbBilletEmis.Text = m_NombreTotalBillet.ToString();
    txtNbBilletTaxe.Text = m_NombreBilletTaxe.ToString();
    txtNbBilletNonTaxesText = m_NombreBilletNonTaxe.ToString();
    txtTotalTaxePercu.Text = m_TotalTaxePercu.ToString();
    txtTotalArgent.Text = m_TotalArgent.ToString();
}

```

Exemple de Compteurs et d'accumulateurs

Données du client
 Nom du Client: Jacques Siège: 22
 Prix de base du billet: 50 Taxable: t

Emission du billet Total des ventes Fermer

Billet de spectacle
 A: Jacques Numéro du siège: 22
 Prix: 55 \$ Taxe: 3,85 \$ Montant à payer: 58,85 \$

Exemple de Compteurs et d'accumulateurs

Données du client
 Nom du Client: Louise Siège: 12
 Prix de base du billet: 100 Taxable: n

Emission du billet Total des ventes Fermer

Billet de spectacle
 A: Louise Numéro du siège: 12
 Prix: 110 \$ Taxe: 0 \$ Montant à payer: 110 \$

Exemple de Compteurs et d'accumulateurs

Données du client
 Nom du Client: Pierre Siège: 49
 Prix de base du billet: 122,77 Taxable: t

Emission du billet Total des ventes Fermer

Billet de spectacle
 A: Pierre Numéro du siège: 49
 Prix: 122,77 \$ Taxe: 8,59 \$ Montant à payer: 131,36 \$

Exemple de Compteurs et d'accumulateurs

Fermer

Total des Ventes
 Nombre billets émis: 3 Nombre de billets taxés: 2 Nombre billets non taxés: 1
 Montant de taxes perçues: 12,44 \$ Total en caisse: 300,21 \$

5.5 Une nouvelle classe d'un formulaire dans un projet Visual C#

```
// Le fichier Program.cs
namespace Exemple3
{
    static class Program
    {
        /// <summary>
        /// Point d'entrée principal de l'application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmExemple3()); // création de l'objet de votre application
        }
    }
}
```

Pour l'instant, nous n'avons pas à toucher à ce fichier. Visual Studio le crée et le met à jour automatiquement.

Le fichier **.Designer.cs** contient la définition de la partie visuelle de la classe de votre formulaire. Il est créé et édité par Visual Studio lorsque vous concevez votre formulaire :

```
namespace Exemple3
{
    partial class frmExemple3
    {
        private System.ComponentModel.IContainer components = null;
        /// <summary>
        /// Nettoyage des ressources utilisées.
        /// </summary>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Code généré par le Concepteur Windows Form

        /// <summary>
        /// Méthode requise pour la prise en charge du concepteur - ne modifiez pas
        /// le contenu de cette méthode avec l'éditeur de code.
        /// </summary>
        private void InitializeComponent()
        {
            this.btnExecuter = new System.Windows.Forms.Button();
            this.btnAfficher = new System.Windows.Forms.Button();
            this.txtValeur = new System.Windows.Forms.TextBox();
            this.SuspendLayout();
            //
            // btnExecuter
            //
            this.btnExecuter.Location = new System.Drawing.Point(61, 24);
            this.btnExecuter.Name = "btnExecuter";
            this.btnExecuter.Size = new System.Drawing.Size(96, 23);
            this.btnExecuter.TabIndex = 0;
            this.btnExecuter.Text = "Exécuter";
            this.btnExecuter.UseVisualStyleBackColor = true;
            this.btnExecuter.Click += new System.EventHandler(this.btnExecuter_Click);
        }

        #endregion
    }
}
```

```

//
// btnAfficher
//
this.btnAfficher.Location = new System.Drawing.Point(61, 53);
this.btnAfficher.Name = "btnAfficher";
this.btnAfficher.Size = new System.Drawing.Size(96, 23);
this.btnAfficher.TabIndex = 1;
this.btnAfficher.Text = "Afficher";
this.btnAfficher.UseVisualStyleBackColor = true;
this.btnAfficher.Click += new System.EventHandler(this.btnAfficher_Click);
//
// txtValeur
//
this.txtValeur.Location = new System.Drawing.Point(61, 101);
this.txtValeur.Name = "txtValeur";
this.txtValeur.Size = new System.Drawing.Size(100, 20);
this.txtValeur.TabIndex = 2;
//
// frmExemple3
//
this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(242, 273);
this.Controls.Add(this.txtValeur);
this.Controls.Add(this.btnAfficher);
this.Controls.Add(this.btnExecuter);
this.Name = "frmExemple3";
this.Text = "Exemple Variables Membres";
this.ResumeLayout(false);
this.PerformLayout();

}
#endregion

private System.Windows.Forms.Button btnExecuter;
private System.Windows.Forms.Button btnAfficher;
private System.Windows.Forms.TextBox txtValeur;
}

```

Pour l'instant, nous n'avons pas à toucher à ce fichier. Visual Studio le crée et le met à jour automatiquement.

Remarquer les trois (3) dernières déclarations. On a ici la déclaration de trois (3) contrôles qui sont en réalité trois (3) objets de type classe Button et TextBox. Ces trois (3) objets sont membres de la classe frmExemple3 donc accessibles par toutes les méthodes membres de frmExemple3.

Ces trois (3) objets sont initialisés dans la méthode InitializeComponent de la classe frmExemple3.

Le fichier **frmExemple3.cs** contient le code personnalisé des méthodes membres de votre classe. C'est ce fichier que vous éditez :

```
namespace Exemple3
{
    public partial class frmExemple3 : Form
    {
        int m_Compteur;
        double m_Accumulateur;

        public frmExemple3() // le constructeur porte le même nom que la classe
        {
            InitializeComponent();

            m_Accumulateur = 0;
            m_Compteur = 0;

            private void btnExecuter_Click(object sender, EventArgs e)
            {
                string Affichage;
                double Valeur;
                int a = 0;
                double b = 0;
                bool Resultat;

                Resultat = Double.TryParse(txtValeur.Text, out Valeur);
                if(Resultat == false)
                {
                    MessageBox.Show("Entrez un nombre réel");
                    return;
                }
                a = a + 1;
                m_Compteur = m_Compteur + 1; // utilisation d'une variable membre
                b = b + Valeur;
                m_Accumulateur = m_Accumulateur + Valeur;
                Affichage = "Valeur de la variable locale a: " + a.ToString() +
                    "Valeur de la variable locale b: " + b.ToString();
                MessageBox.Show(Affichage);
            }

            private void btnAfficher_Click(object sender, EventArgs e)
            {
                string Affichage;
                int a = 0;
                double b = 0;

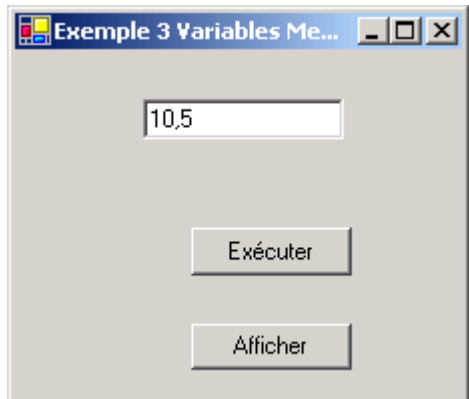
                Affichage = "Nombre de fois: " + m_Compteur.ToString() +
                    "Total: " + m_Accumulateur.ToString();
                MessageBox.Show(Affichage);
                Affichage = "Valeur de la variable locale a: " + a.ToString() +
                    "Valeur de la variable locale b: " + b.ToString();
                MessageBox.Show(Affichage);
            }
        }
    }
}
```

Déclaration des variables membres
Leur nom débute par un m_

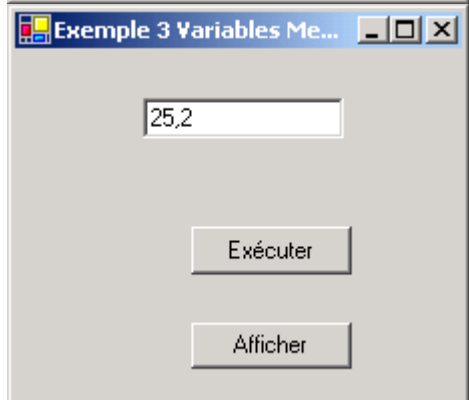
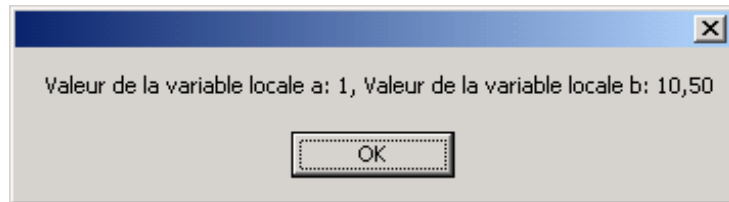
Initialisation des variables membres
Le nom préfixé de m_

Variables locales à la méthode:
btnExecuter_Click

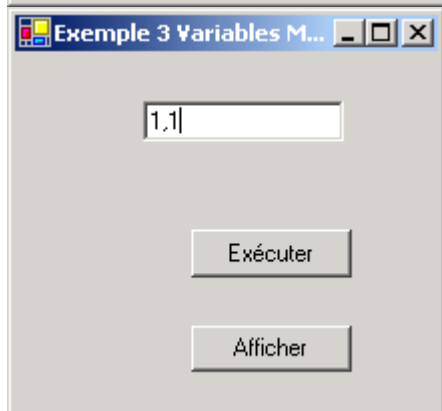
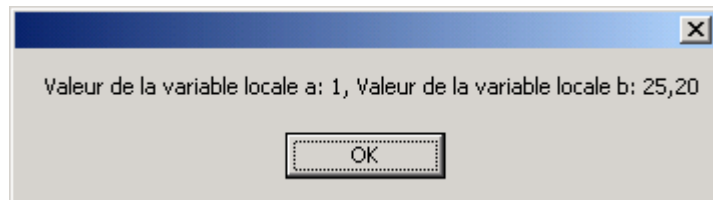
Variables locales à la méthode:
btnAfficher_Click



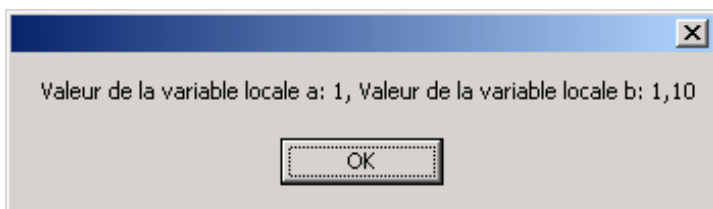
Première
exécution



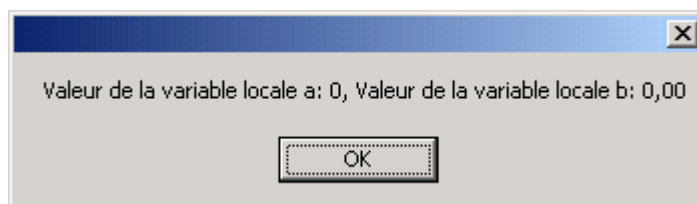
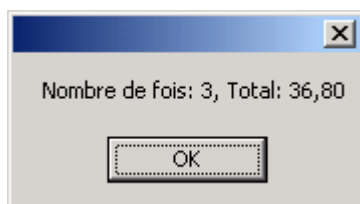
Deuxième
exécution



Troisième
exécution



Exécution
du bouton
Afficher



5.6 Le quiz de révision

Nous désirons établir un système de facturation pour une compagnie d'huile à chauffage. Voici les tarifs exigés: le client paiera 0,25 \$ le litre pour une quantité inférieure à 500 litres. Pour les autres, il paiera 120,00 \$ pour les 500 premiers litres et 0,23 \$ pour les litres restants.

On accorde une réduction de 2 % pour les camions de livraison de type 4.

Il faudra additionner une taxe de vente au montant de la facture dépendant du type de camion de livraison. On aura une taxe de 8% pour les camions de type L, pour les autres, une taxe de 9 %.

Vous disposez à l'entrée des données suivantes:

- Le nom du client (**ne pas valider**)
- Le nombre de litres achetés (**à valider, doit être supérieur à 100**)
- Le code de type de camion: (**à valider**)
 - L - Lourd
 - 4 – 4X4
 - C - camionnette

En appuyant sur le bouton Facturer : obtenir le montant de réduction, le montant de la taxe ainsi que le montant total à payer.

En appuyant sur le bouton Fermeture : afficher la moyenne des litres livrés par une camionnette ainsi que le pourcentage des livraisons faites avec les 4 X 4.

Faire l'analyse, la conception visuelle, l'algorithme constructeur (initialisation des variables membres), l'algorithme de détail (bouton facturation) et l'algorithme de fin (bouton Fermeture).

6 Structure alternatives simple, multiple et imbriquée

6.1 L'utilisation d'une expression simple et multiple

Règle

Lorsque 2 ou plusieurs valeurs d'une même variable déterminent l'ensemble des opérations à effectuer pour un cas précis.

Ex.1 **Si** Age < 18 ou Age > 65
 Rabais ← Prix / 2
 Sinon
 Rabais ← Prix / 3

Ex.2 **Si** Salaire ≥ 10000 et Salaire < 15000 ou Salaire > 100000

6.2 Le Si imbriqué

On est en présence d'un SI IMBRIQUÉ, lorsqu'une action du bloc Vrai ou du bloc Faux est un autre SI.

Soient les algorithmes suivants:

Lis Vitesse Si Vitesse < 50 Amende ← 0 Si Vitesse ≥ 50 ET Vitesse < 70 Amende ← (Vitesse - 50) * 2 Si Vitesse ≥ 70 ET Vitesse < 90 Amende ← (Vitesse - 50) * 3 Si Vitesse ≥ 90 Amende ← (Vitesse - 50) * 4 Écris Amende	Lis Vitesse Si Vitesse < 50 Amende ← 0 Sinon Si Vitesse < 70 Amende ← (Vitesse - 50) * 2 Sinon Si Vitesse < 90 Amende ← (Vitesse - 50) * 3 Sinon Amende ← (Vitesse - 50) * 4 Écris Amende
--	--

Quelles constatations peut-on faire en suivant ces 2 algorithmes.

6.3 L'utilisation des Si imbriqués

On tend d'abord à résoudre les problèmes par des **Si** à expressions complexes.

Lorsque les valeurs d'une ou plusieurs variables qui déterminent les différents traitements à effectuer sont mutuellement exclusives (c'est-à-dire l'une exclut les autres).

Règle

Lorsque d'une part deux (2) ou plusieurs valeurs de variables différentes déterminent l'ensemble des instructions à effectuer pour un cas précis et d'autre part, qu'on ne génère pas de répétition de même condition.

Exemple de répétition de conditions.

```
Si A = 1 et B = 7 et C < 5
    Message ← "AAAA"
Si A = 1 et B = 7 et C >= 5
    Message ← "BBBB"
```

Comment devrait-on l'écrire de façon plus efficace?

```
Si A = 1 et B = 7
    Si C < 5
        Message ← "AAAA"
    Sinon
        Message ← "BBBB"
```

Exemple avec une variable

```
Si A = 1
    B ← B * 2
    C ← C + D
Sinon
    Si A = 2
        B ← B * 3
        C ← C - D
    Sinon
        B ← B * 4
        C ← D
```

Exemple avec plus d'une variable

```
Si F = 1 ET G < 0
    Nombre ← 11
Sinon
    Si F = 2 ET G > 10
        Nombre ← 22
    Sinon
        Si F < 35 ET G = 3
            Nombre ← 33
```

Lorsque deux (2) ou plusieurs valeurs de variables différentes déterminent l'ensemble des opérations à effectuer pour un cas précis et qu'on ne peut résoudre le problème par des SI en série avec expression complexe qu'en répétant des mêmes conditions.

Exemple à ne pas faire.

```
Si CategorieEtudiant = 0 et Bourse > 5000
    Message ← "Tu es chanceux"
Si CategorieEtudiant = 0 et Bourse <= 5000
    Message ← "Tu dois te serrer la ceinture"
Si CategorieEtudiant < > 0
    Message ← "Tu n'as pas droit à la bourse"
Écris Message
```

Pourquoi?

Utiliser les Si imbriqués

```
Si CategorieEtudiant = 0
    Si Bourse > 5000
        Message ← "Tu es chanceux"
    Sinon
        Message ← "Tu dois te serrer la ceinture"
Sinon
    Message ← "Tu n'as pas droit à la bourse"
Écris Message
```

Pourquoi c'est mieux?

6.4 Les étapes à suivre pour résoudre une structure alternative

1. Résoudre par l'utilisation de SI complexe si nécessaire.
2. Si les cas sont mutuellement exclusifs alors imbriquer.
3. Si des répétitions de conditions sont générées, alors imbriquer.

6.5 Le quiz de révision

- a) Écrire l'algorithme pour calculer le montant de pénalité pour un abonné remettant un volume en retard. Les informations suivantes sont fournies:

Le nombre de jours de retard
L'âge de l'abonné

Afficher le montant de pénalité à déboursier sachant:

- aucune pénalité pour les personnes âgées de 65 ans et plus, et qui ont 2 jours et moins de retard
- pénalité de 1.00\$ + 0.05\$ par jour de retard pour les personnes âgées de 5 à 64 ans inclus et qui ont 7 jours de retard et plus
- pénalité de 1.00\$ + 0.03\$ par jour de retard pour tous les autres cas.
- un maximum de 2.00\$ de pénalité est chargé dans tous les cas.

- b) Écrire l'algorithme qui lit la vitesse d'un véhicule au moment d'une infraction ainsi le nombre de fois que le conducteur a été en infraction et qui affiche le montant d'infraction sachant que:

Montant initial	=	0	pour chaque	KM \leq 50	
	=	+ 1\$	"	"	KM $>$ 50 et \leq 70
	=	+ 2\$	"	"	KM $>$ 70 et \leq 90
	=	+ 3\$	"	"	KM $>$ 90

Puis montant infraction = montant initial si 1^{re} infraction
= 2 fois le montant initial si 2^e infraction
= 3 fois le montant initial si 3^e infraction ou plus

Ex. Si Vitesse 73 KM et 2^e infraction:
Il y a 20 km/h à 1\$ plus 3 km/h à 2\$ qui donne 20\$ + 6\$ = 26\$ et comme c'est une deuxième infraction 26\$ * 2 pour un total de 52\$

Ex. Si Vitesse 93 KM et 1^{ère} infraction:
Il y a 20 km/h à 1\$ plus 20 km/h à 2\$ et 3 km/h à 3\$ qui donne 20\$ + 40 + 9\$ = 69\$ et comme c'est une première infraction 69\$ * 1 pour un total de 69\$

- c) Le club vidéo "Bon film" veut informatiser sa facturation. Présentement, il possède 4 catégories de film:

NO-FILM	CATÉGORIE
1-50	Enfant
100-200	Humour
201-275	Sentimental
500-580	Aventures

À chaque fois qu'une personne retourne le film emprunté on obtient l'information suivante:
no membre ou 0 si le client n'est pas membre
no film
nombre d'heures empruntées

Le coût de location se calcule de la façon suivante:

2.00\$ de base pour tous les genres de film

plus

0.10\$ l'heure pour la catégorie "enfant"

1.50\$ + 0.05\$ l'heure pour les catégories "humour et aventures"

1.25\$ + 0.05\$ l'heure pour la catégorie sentimental

plus

1.00\$ pour ceux qui n'ont pas de numéro de membre.

Une pénalité de 0.25\$ l'heure est chargée pour chaque heure dépassant 24 heures de location.

Exemples: 26 heures = pénalité de 0,50\$

30 heures = pénalité de 1.50\$

Écrire l'algorithme qui affichera le coût de location, la pénalité et le coût total.

NOTE: Valider le numéro de film facilitera grandement le traitement.

- d) Lire un no de client, l'âge, le statut civil (C = célibataire, M = marié) et le nombre d'accidents et afficher la prime d'assurance à payer.

Prime = 300.00\$

Plus

5% de 300.00\$ pour les célibataires

5% de 300.00\$ pour les personnes de 21 ans et moins

5% de 300.00\$ pour les personnes de 65 ans et plus

25.00\$ par accident.

Ne pas oublier de valider le statut civil.

- e) Écrire l'algorithme pour résoudre le problème suivant:

La compagnie d'aviation Vol-Air veut à partir:

du numéro de vol
du nombre de réservations de passagers
de la capacité (nombre de sièges disponibles à la clientèle).

L'affichage demandé est le suivant:

Le nombre de places encore disponibles

Et une mention:

"vide" s'il y a moins de 25% d'occupation de l'avion

"moitié plein" s'il y a moins de 75% d'occupation mais au moins 25% et plus

"plein" s'il y a au moins 75% d'occupation

f) Écrire l'algorithme pour résoudre le problème suivant:

Je veux calculer et afficher le salaire annuel de mes employés d'après l'expérience et la scolarité. Comme données, j'ai le numéro de l'employé, sa scolarité et son expérience.

Le salaire se calcule comme suit:

- Salaire annuel =
- a) 15000 pour scolarité = 15 et expérience = 0
 - b) 15000 pour scolarité < 11
 - c) 18000 pour scolarité = 20 + 1000\$ pour chaque année d'expérience > 5
 - d) 15000 pour scolarité = 11 et expérience > = 5
 - e) 10000 + scolarité * 150 + expérience * 500 pour tous les autres cas.

g) Faire la conception visuelle et écrire l'algorithme.

La société Distincte vous demande d'écrire l'algorithme pour son système de perception d'impôt, sachant que pour chaque payeur de taxes, on a les informations suivantes:

Le salaire annuel
Le nombre d'années de travail consécutives
Le statut matrimonial: Célibataire (C)
Marié (M)
Veuf (V)
Divorcé(D)
L'âge
Le nombre d'emplois occupé pendant l'année.
Le nombre d'enfants

Ne validez que le statut matrimonial.

L'impôt de base est de 5000 \$ plus 500 \$ pour chaque année de travail consécutive.

PLUS: (les % se calculent à partir de l'impôt de base et une personne peut en avoir plusieurs)

- 10 % pour les plus de 40 ans inclus et moins de 55 ans inclus.
- 5 % pour les plus de 55 ans.
- 15 % pour les contribuables ayant occupés plus de 2 emplois pendant l'année.
- 1000 \$ s'il n'a pas d'enfant

MOINS: (les % se calculent à partir de l'impôt de base et une personne peut en avoir plusieurs)

- 3 % pour un veuf et qui a 2 enfants et plus
- 5 % pour un célibataire avec au moins 1 enfant
- 1.5 % pour les autres statuts ayant au moins 1 enfant
- 1 % pour les moins de 40 ans.

Aucun impôt ne devrait excéder 75 % du salaire.

Note: Une personne peut être célibataire et avoir des enfants.

Afficher le montant d'impôt à payer.

h) Faire l'analyse, la conception visuelle, les algorithmes, construire un jeu d'essais complet et faire la trace avec 2 essais de votre jeu d'essais.

Lire les dossiers d'électeurs qui contiennent pour chacun:

- Sexe (F ou M) à valider
- Nom
- Âge
- Salaire annuel
- État civil (ne pas valider M = Marié, C = Célibataire, A = Autre)

On veut faire afficher le salaire hebdomadaire, le nombre d'électeurs féminins, le nombre d'électeurs masculins, le nombre d'électeurs féminins qui gagnent plus de 30 000 \$ par année, le total des salaires annuels des hommes, la moyenne des salaires hebdomadaires des personnes de 50 ans et plus, le pourcentage des hommes qui gagnent moins de 50 000 \$ et l'état civil en lettre.

Note: Distinguez bien les valeurs à afficher par électeur des valeurs à afficher pour l'ensemble des électeurs.

7 Structure répétitive

7.1 Définition

La structure répétitive permet d'exécuter plusieurs fois une séquence d'instructions.

Synonyme : Boucle, itération.

Les instructions à exécuter peuvent être des structures séquentielles, alternatives et/ou d'autres structures répétitives.

7.2 Des exemples tirés de la vie courante

1- Clouer un clou dans une planche

prendre en main le marteau
prendre en main un clou
placer le clou sur la planche
Tant que le clou n'est pas enfoncé dans la planche
 donner un coup de marteau sur le clou
déposer le marteau

2- Mettre la table pour 4 personnes

placer la nappe sur la table
Pour NombreFois = 1 à 4
 placer une assiette sur la table
 placer ustensile sur la table
 placer un verre sur la table
s'asseoir à table pour manger

3- Boire un verre de limonade

prendre un verre
remplir le verre de limonade
Faire
 prendre une gorgée
 attendre un peu
Tant que le verre n'est pas vide et que j'ai encore soif
déposer le verre
s'essuyer la bouche

7.3 La structure répétitive Tant que

La structure répétitive **Tant que** Consiste à faire répéter une ou plusieurs instructions tant qu'une expression est vraie. L'expression est évaluée en premier, si le résultat est Vrai, la ou les instructions sont exécutée(s). Ensuite, le traitement retourne à l'entête du **Tant Que** et l'expression est évaluée à nouveau. Si le résultat est vrai le traitement est exécuté encore une fois et le processus continue. Si l'expression est fausse le traitement passe à l'instruction qui suit le **Tant Que**.

Format: **Tant que** expression
 instruction
 [instruction]

L'expression suit les mêmes règles que dans une structure alternative. Elle peut être composée d'une condition simple ou de plusieurs conditions simples reliées par un opérateur logique. Les instructions contenues dans le **Tant que** sont décalées d'une tabulation par rapport à l'entête du **Tant que**.

La ou les variables utilisée(s) dans l'expression doit obligatoirement recevoir une valeur avant le **Tant que** puisque l'expression est évaluée en premier.

La ou les variables utilisée(s) dans l'expression doit obligatoirement changer de valeur dans le bloc du Tant que (sinon on aura une boucle infinie).

Si l'expression est FAUSSE à la première évaluation du **Tant que**, la ou les instructions ne seront jamais exécutées.

Exemple: Faire la somme des nombres de 1 à 500 et faire afficher cette somme.

```
Somme ← 0
Nombre ← 1
Tant que Nombre ≤ 500
    Somme ← Somme + Nombre
    Nombre ← Nombre + 1
Écris Somme
```

7.4 La traduction en C# du Tant Que

```
while (expression)
{
    instruction;
    [instruction;]
    ....
}
```

Note: comme pour l'instruction **if**, il n'y a pas de point-virgule (;) après l'expression et l'expression est toujours entre parenthèses. Les instructions dans le bloc à répéter sont décalées d'une tabulation par rapport à l'entête. Si le bloc à répéter ne contient qu'une instruction, les accolades sont optionnelles.

7.5 La structure répétitive Faire...Tant que

La structure répétitive **Faire...Tant que** consiste à faire répéter une ou plusieurs instructions tant qu'une expression est vraie. Au contraire du **Tant que**, la ou les instructions sont exécutée(s) une première fois, ensuite l'expression est évaluée, si le résultat est Vrai, la ou les instructions sont exécutée(s) à nouveau et l'expression est évaluée à nouveau. Si le résultat est vrai le traitement est exécuté encore une fois et le processus continue. Si l'expression est fausse le traitement passe à l'instruction qui suit le **Tant que**.

Format:

```
Faire
    instruction
    [instruction]
    .....
Tant que expression
```

L'expression suit les mêmes règles que dans une structure alternative. Elle peut être composée d'une condition simple ou de plusieurs conditions simples reliées par un opérateur logique.

Les instructions contenues dans le bloc sont décalées par rapport à l'entête de la boucle.

La ou les **variables utilisée(s) dans l'expression doit obligatoirement recevoir une valeur avant** l'évaluation de l'expression avant ou dans le bloc.

La ou les **variables utilisée(s) dans l'expression doit obligatoirement changer de valeur dans le bloc du Tant que** (sinon on aura une boucle infinie).

Les instructions contenues dans le bloc de la boucle sont toujours exécutées au moins une fois puisque l'expression est évaluée après l'exécution du corps de la boucle.

Exemple: Faire la somme des nombres de 1 à 500 et faire afficher cette somme.

```
Somme ← 0
Nombre ← 1
Faire
    Somme ← Somme + Nombre
    Nombre ← Nombre + 1
Tant que nombre ≤ 500
    Écris Somme
```

7.6 La traduction en C# du Faire...Tant Que

```
do
{
    instruction;
    [instruction;]
    ....
}
while (expression);
```

Remarquez le point-virgule (;) après l'expression. Pas de ; après le do

7.7 La structure répétitive Pour

La structure répétitive **Pour** consiste à faire répéter une ou plusieurs instructions un nombre **connu** de fois. La **variable qui contrôle** la boucle reçoit une **valeur initiale** dans l'entête de la boucle. La **condition** d'exécution portant sur la variable de contrôle est aussi **indiquée dans l'entête** de la boucle ainsi que **l'instruction d'incrément** (valeur qui modifiera la variable de contrôle à chaque répétition). Cette instruction doit être utilisée lorsque le nombre de fois que la boucle se fera est connu avant de la commencer. Il est interdit de quitter un **Pour** en jouant avec la variable de contrôle dans les instructions du bloc **Pour** ou en utilisant des instructions de brisure (break).

Format:

```
Pour variable_de_contrôle ← valeur de départ, condition, incrément  
    instruction  
    [instruction]  
    ...
```

Action:

- 1) La variable de contrôle est initialisée à la valeur de départ.
- 2) Tant que la condition est vraie on exécute le bloc d'instructions, puis, la variable de contrôle est ajustée par la valeur de l'incrément.
- 3) Si la condition est fausse le contrôle passe à l'instruction qui suit le **Pour**.

Remarques: Les valeurs initiales, valeurs finales et incréments peuvent être des constantes, des variables ou des expressions arithmétiques.

Exemples: **Pour** $i \leftarrow 1$; $i \leq 5$; $i \leftarrow i+1$

Pour i qui vaut 1, tant que i est plus petit ou égal à 5 alors i vaut $i + 1$

Pour $i \leftarrow a$; $i \leq b$; $i \leftarrow i+2$

Pour i qui vaut a , tant que i est plus petit ou égal à b alors i vaut $i + 2$

Pour $i \leftarrow 1$; $i \leq n+3$; $i \leftarrow i+1$

Pour i qui vaut 1, tant que i est plus petit ou égal à $n+3$ alors i vaut $i + 1$

Pour $i \leftarrow n$; $i \geq 1$; $i \leftarrow i-1$

Pour i qui vaut n , tant que i est plus grand ou égal à 1 alors i vaut $i - 1$

Exemple: Faire la somme des nombres de 1 à 500 et faire afficher cette somme.

```
Somme ← 0
Pour Nombre ← 1; Nombre ≤ 500; Nombre ← Nombre + 1
    Somme ← Somme + Nombre
Écris Somme
```

Exercices: Qu'affichent ces algorithmes?

- 1)

```
Somme ← 0
Nombre ← 2
Pour Compteur ← Nombre*Nombre; Compteur ≤ Nombre*Nombre+2; Compteur ← Compteur+1
    Somme ← Somme + Compteur
Écris Somme, Compteur, Nombre
```
- 2)

```
Somme ← 0
Nombre ← 2
Pour Compteur ← Nombre * Nombre + 2; Compteur ≥ Nombre; Compteur ← Compteur -1
    Somme ← Somme + Compteur
Écris Somme, Compteur, Nombre
```
- 3)

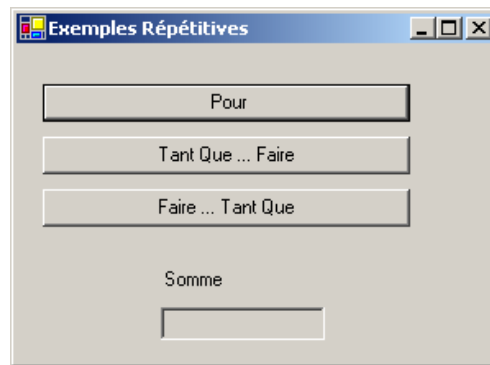
```
Somme ← 0
Nombre ← 2
Pour Compteur ← Nombre; Compteur > Nombre * Nombre ; Compteur ← Compteur -1
    Somme ← Somme + Nombre
Écris Somme, Nombre
```

7.8 Traduction en C# du Pour

```
for (initialisation; condition de fin ; incrémentation)
{
    instruction;
    instruction;
    ....
}
```

Aucune instruction du bloc d'un for ne peut être un return, un break ni une instruction qui modifie la variable de contrôle.

7.9 Des exemples



```
private void btnPour_Click(object sender, System.EventArgs e)
{
    int Somme=0;
    int Compteur;
    for (Compteur = 1; Compteur <=5; Compteur = Compteur + 1)
        Somme = Somme + Compteur;
    txtSomme.Text = Somme.ToString();
}

private void btnTantQueFaire_Click(object sender, System.EventArgs e)
{
    int Somme=0;
    int Compteur=1;
    while (Compteur <=5)
    {
        Somme = Somme + Compteur;
        Compteur = Compteur + 1;
    }
    txtSomme.Text = Somme.ToString();
}

private void btnFaireTantQue_Click(object sender, System.EventArgs e)
{
    int Somme=0;
    int Compteur=1;
    do
    {
        Somme = Somme + Compteur;
        Compteur = Compteur + 1;
    }
    while (Compteur <=5);
    txtSomme.Text = Somme.ToString();
}
```

7.10 Les boucles imbriquées

Il est possible qu'une boucle soit totalement comprise dans une autre boucle. Les boucles construites de cette façon sont appelées BOUCLES IMBRIQUÉES.

Les règles s'appliquant aux boucles imbriquées sont les mêmes que celles s'appliquant aux boucles simples. Cependant, il y a un point très important: **les boucles ne doivent pas se chevaucher.**

On utilise les boucles imbriquées lorsqu'un traitement doit se répéter plus d'une fois à l'intérieur d'une tâche qui elle-même doit se répéter. Par exemple on met la table 3 fois dans une journée et chaque fois, on doit répéter les mêmes actions pour dresser 4 couverts.

Faire la trace de ces algorithmes. Remarquez que chaque boucle est contrôlée par une variable différente.

a) La valeur lue est: 4

```
Lis Code
Pour Ctr1  $\leftarrow$  1; Ctr1  $\leq$  Code; Ctr1  $\leftarrow$  Ctr1+1
    Écris "Début Boucle-1", Ctr1
    Ctr2  $\leftarrow$  1
    Tant que Ctr2  $\leq$  2
        Écris "Boucle-2", Ctr2
        Ctr2  $\leftarrow$  Ctr2 + 1
    Écris "Fin Boucle 1", Ctr1
Écris "Fin du traitement"
```

b) Les valeurs lues sont: 4, 2, 3

```
Lis Code1, Code2, Code3
Pour Ind1  $\leftarrow$  1 ; Ind1  $\leq$  Code1; Ind1  $\leftarrow$  Ind1+1
    Écris "Début Boucle-1", Ind1
    Pour Ind2  $\leftarrow$  1; Ind2  $\leq$  Code2; Ind2  $\leftarrow$  Ind2 + 1
        Écris "Début Boucle-2", Ind2
        Pour Ind3  $\leftarrow$  1; Ind3  $\leq$  Code3; Ind3  $\leftarrow$  Ind3 + 1
            Écris "Début Boucle-3", Ind3
        Écris "Fin Boucle 2", Ind2
    Écris "Fin Boucle 1", Ind1
Écris "Fin du traitement"
```

c) Les valeurs lues sont: 10, 12, 23

```
Lis Code1, Code2, Code3
Pour Ind1 ← 13; Ind1 >= Code1; Ind1 ← Ind1-1
    Écris "Début Boucle-1", Ind1
    Pour Ind2 ← 8; Ind2 <= Code2; Ind2 ← Ind2+2
        Écris "Début Boucle-2", Ind2
        Pour Ind3 ← 25; Ind3 >= Code3; Ind3 ← Ind3 - 1
            Écris "Début Boucle-3", Ind3
        Écris "Fin Boucle 2", Ind2
    Écris "Fin Boucle 1", Ind1
Écris "Fin du traitement"
```

7.11 Le quiz de révision

Faire l'algorithme pour chaque énoncé en utilisant en premier un **Tant que**, en deuxième un **Faire Tant que** et enfin un **Pour** si applicable.

1. Faire la somme des nombres 1 à 100 et afficher cette somme.
2. Avec une boucle, faire la somme des chiffres impairs de 5 à 69 inclusivement. Afficher cette somme ainsi que le nombre de fois que la boucle s'est exécutée.
3. Calculer et afficher le factoriel d'un nombre entier positif lu au clavier. Exemple, le factoriel de 7 se calcule comme suit: $1*2*3*4*5*6*7$
4. Calculer et afficher la puissance entière d'un nombre entier lu au clavier. On fournit 2 nombres entiers positifs (disons X et N) en entrée et on veut obtenir la valeur de X exposant N. **Vous procéderez par une méthode de multiplications successives.**

$$X^N = X * X * X * \dots * X * X \text{ (N fois).}$$

5. Cumuler les nombres à partir de 5 par intervalle de 3 jusqu'à ce que cette somme dépasse 40. Afficher cette somme ainsi que le nombre de fois que la boucle s'est exécutée.
6. Calculer et afficher le produit de deux (2) nombres entiers (disons A et B) lus au clavier. **Vous devez utiliser une méthode impliquant des additions successives.**

$$A * B = A + A + A + \dots + A \text{ (B fois).}$$

7. Faire la table de multiplication du 5 pour obtenir les résultats suivants:

$$\begin{aligned} 1 \times 5 &= 5 \\ 2 \times 5 &= 10 \\ 3 \times 5 &= 15 \\ &\dots \\ 10 \times 5 &= 50 \end{aligned}$$

8. Une personne fait un dépôt initial de 0.01 \$ et tous les jours, elle double ce dépôt, et ce, pendant 30 jours. Afficher le numéro du jour, le dépôt et le solde de chaque journée.
9. Une personne fait un dépôt initial de 0.01 \$ et tous les jours, elle double ce dépôt. Afficher le montant d'argent exact économisé lorsqu'elle aura atteint les 1000 \$ et le nombre de jours qu'elle a pris pour obtenir ce montant.
10. Afficher la table de multiplication de 1 à 12 pour un chiffre entré au clavier. Le chiffre lu doit être entre 1 et 12 inclusivement.
11. Sachant que la méthode Hasard choisit un nombre entier au hasard, afficher un nombre au hasard se situant entre 0 et 15 inclusivement en utilisant cette méthode comme suit :

NombreHasard ← Hasard()

12. On place un montant de 1000 \$ au début de chaque année à 10 % d'intérêt composé annuellement et ce pendant 10 ans. Afficher le numéro de l'année, le montant cumulatif des dépôts, l'intérêt calculé à la fin de l'année sur le solde et le nouveau solde après l'ajout de l'intérêt.

1	1000 \$	100 \$	1100 \$
2	2000 \$	210 \$	2310 \$
3	3000 \$	331 \$	3641 \$
....			

13. Évaluer et afficher la somme des 30 premiers termes de:

a. $1 + 2 + 4 + 8 + 16 + 32 + \dots$

b. $1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{6} + \frac{1}{8} - \frac{1}{10} + \frac{1}{12} \dots$

c. $1 + 4 + 7 + 10 + 13 + 16 + 19 \dots$

14. Un nombre est dit premier s'il n'est divisible que par 1 et par lui-même. Afficher un message significatif si un nombre lu est premier.
15. Lire une phrase au clavier ainsi qu'une lettre. Afficher le nombre de fois que cette lettre apparaît dans la phrase.
16. Lire une phrase au clavier ainsi qu'une lettre. Afficher à quelle position dans la phrase la lettre apparaît pour la première fois.
17. Afficher un message significatif si une chaîne de caractères lue est un palindrome. Rappel: un palindrome est une chaîne dont la lecture de gauche à droite donne le même résultat que de droite à gauche. Par exemple: LAVAL, ELU PAR CETTE CRAPULE
18. Afficher la séquence des heures et des minutes d'une journée de 00:00 à 23:59.

19. Afficher la séquence des heures et des minutes d'une journée par intervalle de 15 minutes de 00:00 à 23:59.
20. Afficher un losange formé d'un caractère fourni en entrée. La largeur du losange est aussi fournie en entrée. Si la largeur est paire, on doit l'augmenter de 1 et faire afficher le losange.
21. Afficher le nombre de mots dans un texte où chaque mot est séparé du suivant par exactement un espace. La fin du texte lue est indiqué par le caractère point (.).
22. Afficher le nombre de mots dans un texte où chaque mot est séparé du suivant par au moins un espace. La fin du texte lue est indiqué par le caractère point (.).

8 Tableaux

8.1 Définition

Un tableau est une séquence de données de même type portant un seul nom. L'accès à un élément individuel d'un tableau se fait en utilisant sa position dans le tableau qu'on appelle indice ou index. Les éléments d'un tableau sont situés dans de la mémoire contiguë. Un programme peut accéder aussi rapidement à tous les éléments d'un tableau.

8.2 Les tableaux à une dimension

Autre nom: Vecteur.

On peut le représenter comme ceci :

0	1	2	3

Pour utiliser un tableau de quatre (4) éléments représentant des nombres entiers:

Déclaration obligatoire : MonTableau[4]

Affectation de chaque élément:

MonTableau[0] ← 5
MonTableau[1] ← 10
MonTableau[2] ← 12
MonTableau[3] ← 8

5	10	12	8
0	1	2	3

Pour accéder à un élément d'un tableau: Variable ← MonTableau [Indice]

Indice peut être une constante, une variable ou une expression entière positive.

Nombre ← MonTableau[2]

La variable Nombre contiendra le nombre 12 en utilisant le tableau de l'exemple précédent.

Pour changer le contenu d'un élément: MonTableau [indice] ← nouvelleValeur

Indice peut être une constante, une variable ou une expression entière positive.

MonTableau[1] ← 43

Le tableau ci haut devient alors :

5	43	12	8
0	1	2	3

Pour initialiser un tableau à sa déclaration : MonTableau[4] ← {5,10,12,8}

C'est possible seulement lors de sa déclaration dans l'algorithme.

Attention! Il est **impossible** de changer la dimension d'un tableau après sa déclaration. On doit donc prévoir suffisamment d'espace pour toutes les données.

Attention! Accéder à un indice de tableau à l'extérieur de celui-ci correspond à une erreur de programmation et peut donner lieu à des erreurs d'exécution.

MonTableau[4] \leftarrow {5,10,12,8}

Variable \leftarrow MonTableau[10]

Évidemment, le tableau étant de grandeur 4, la case MonTableau[10] n'existe pas et on ne peut y accéder.

Attention! Le premier indice d'un tableau est toujours 0. Ce qui implique que le dernier indice est le nombre d'éléments moins un.

8.2.1 Cas d'un tableau de caractères

MonTableauCaractere[6]

MonTableauCaractere [0] \leftarrow 'S'

MonTableauCaractere [1] \leftarrow 'a'

MonTableauCaractere [2] \leftarrow 'l'

MonTableauCaractere [3] \leftarrow 'u'

MonTableauCaractere [4] \leftarrow 't'

MonTableauCaractere [5] \leftarrow '!''

Le tableau contient alors :

'S'	'a'	'l'	'u'	't'	'!''
0	1	2	3	4	5

Attention! Une chaîne est en réalité un tableau de caractères.

On aurait donc pu arriver au même résultat de la façon suivante :

MonTableauCaractere \leftarrow "Salut!"

On peut donc accéder à la troisième lettre de la façon suivante :

MaLettre \leftarrow MonTableauCaractere[3]

La variable MaLettre contiendra alors la valeur 'u'.

8.2.2 Cas d'un tableau de chaines

Un tableau de trois (3) chaines de caractères.

MonTableauDeChaines[3]

MonTableauDeChaines[0] ← "Marc";
MonTableauDeChaines[1] ← "Mathieu"
MonTableauDeChaines[2] ← "Pierre"

Le contenu du tableau :

"Marc"	"Mathieu"	"Pierre"
0	1	2

On voit bien que l'on peut faire des tableaux d'autant de dimensions que l'on en a besoins. Cependant, il devient difficile de se les représenter avec une image.

8.3 La traduction en C# des tableaux à une dimension

8.3.1 La déclaration

`type [] NomDuTableau = new type[NombreÉléments]`
OU { une suite des valeurs séparées par une virgule};

Exemples :

```
char [ ]MonTableauCaractere = new char [6];  
int [ ]MonTableauEntier = {5, 10, 12, 8};  
string [ ]MonTableauChaine = new string [6];
```

8.3.2 L'affection

NomDuTableau[indice] = expression;

L'expression doit donner un résultat de même type que le type des éléments du tableau.

Exemples :

```
MonTableauCaractere[4] = 'z';  
MonTableauEntier[1] = 15 * 4;  
MonTableauChaine[2] = "Christian";
```

8.3.3 L'accès à un élément

Variable = NomDuTableau[indice];

On peut utiliser un élément d'un tableau dans toute expression qui nécessite une valeur du type de l'élément.

Exemples :

```
UneLettre = MonTableauCaractere[4];
```

```
UnEntier = MonTableauEntier[1] * 5 ;
```

```
UnNom = MonTableauChaine[2] + MonTableauChaine[4] + " Bonjour";
```

```
MonTableauEntier[2] = MonTableauEntier[1] * 5 ;
```

8.4 Les algorithmes de base de manipulation de tableaux

- a) Le parcours de tous les éléments d'un tableau d'entier pour les additionner et afficher la somme.

```
MonTableauEntier[10] ← {34, 68, 254, 20, 43, 662, 58, 156, 33, 87}
```

```
Somme ← 0
```

```
Indice ← 0
```

```
Tant que Indice < 10
```

```
    Somme ← Somme + MonTableauEntier[Indice]
```

```
    Indice ← Indice + 1
```

```
Écris Somme
```

- b) Le parcours de tous les éléments d'un tableau d'entier pour calculer la moyenne des valeurs et afficher la moyenne.

```
MonTableauEntier[10] ← {34, 68, 254, 20, 43, 662, 58, 156, 33, 87}
```

```
Somme ← 0
```

```
Indice ← 0
```

```
Tant que Indice < 10
```

```
    Somme ← Somme + MonTableauEntier[Indice]
```

```
    Indice ← Indice + 1
```

```
Moyenne ← Somme / 10
```

```
Écris Moyenne
```

- c) Retrouver une chaîne en particulier dans un tableau selon l'indice lu au clavier et afficher la chaîne.

```
TableauMois[12] ← {"Janvier", "Février", "Mars",  
                  "Avril", "Mai", "Juin",  
                  "Juillet", "Août", "Septembre",  
                  "Octobre", "Novembre", "Décembre"}
```

Lis Indice
Si Indice > 11 ou Indice < 0
 Erreur Terminer
Mois ← TableauMois[Indice]
Écris Mois

N.B. **ne jamais utiliser** de **Si** ni même de **switch case** pour résoudre un problème de ce genre.

- d) Retrouver l'indice d'une valeur en particulier dans un tableau de chaîne selon la chaîne lue au clavier et afficher l'indice.

TableauMois[12] ← {"Janvier", "Février", "Mars",
 "Avril", "Mai", "Juin",
 "Juillet", "Août", "Septembre",
 "Octobre", "Novembre", "Décembre" }

Lis NomMois
Indice ← 0
Tant que Indice < 12 ET NomMois <> TableauMois[Indice]
 Indice ← Indice + 1
Si Indice < 12
 Écris Indice
Sinon
 Écris "Pas trouvé"

N.B. **ne jamais utiliser** de **Si** ni même de **switch case** pour résoudre un problème de ce genre.

Particularité du C# et son environnement de développement .NET

8.5 L'instruction SWITCH-CASE

Plusieurs langages modernes nous offrent une instruction facilitant l'écriture de **Si** imbriqués. En C#, cette instruction se nomme Switch – case.

Nous pouvons utiliser cette instruction sous certaines conditions permettant de viser l'efficacité du programme :

- Pas plus de 5 niveaux d'imbrication
- Les valeurs de l'expression dans la condition sont de types simples y compris la chaîne de caractères.
- Ne peut être résolu à l'aide d'un tableau

```
switch (expression entière)
{
    case constante:    instruction;
                      [instruction; ...]
                      break;

    [case constante:  instruction;
      [instruction; ...] ]
      break;

    ...
    [ default :      instruction;
      [instruction; ...] ]
      break;
}
```

Expression entière représente soit une variable entière (int) , caractère (char) , chaîne de caractères (string) ou soit une expression qui donne un résultat entier.

```
ex:    int a;
        char b;
        switch (a)
        switch ( (a + 3) * 2)
        switch (b)
```

L'expression est évaluée et sa valeur est recherchée successivement parmi les valeurs des diverses constantes proposées. En cas d'égalité, les instructions correspondantes sont alors exécutées.

Lorsqu'aucun cas n'est sélectionné, on exécute les instructions correspondant au cas default s'il existe.

L'énoncé default est facultatif.

Ne pas oublier les break après chacun des blocs d'instructions.

Exemples

```
private void btnTraiter_Click(object sender, System.EventArgs e)
{
    int Nombre;
    string EnLettre;
    bool Resultat;
    Resultat = Int32.TryParse(txtChiffre.Text, out Nombre);
    if(Resultat == false)
    {
        MessageBox.Show("Entrez un nombre entier");
        return;
    }
    switch (Nombre)
    {
        case 7:          EnLettre = "SEPT";
                        break;

        case 12:         EnLettre = "DOUZE";
                        break;

        case 15:         EnLettre = "QUINZE";
                        break;

        default: EnLettre= "ni SEPT, ni DOUZE, ni QUINZE";
                        break;
    }
    MessageBox.Show(EnLettre);
}
```

Il est à noter que pour une variable d'un autre type qu'entier, caractère ou chaîne, nous ne pouvons pas utiliser cette instruction. Par conséquent, l'utilisation des **SI IMBRIQUÉS** est indiquée.

```
...
if (Montant < 0.0)
    Valeur = "NEGATIF";
else
    if (Montant == 0.0)
        Valeur = "NUL";
    else
        Valeur = "POSITIF";
```

8.6 L'utilisation des contrôles Cases à Cocher

Le contrôle Case à cocher est un moyen de présenter une option à l'utilisateur; ce dernier peut la cocher pour sélectionner l'option ou supprimer la coche pour annuler la sélection. Lorsque l'utilisateur coche ou décoche une case à cocher, la valeur de la propriété **Checked** change et l'événement **CheckedChanged** se produit. La propriété **Checked** est de **type booléen** et vaut **true si la case est cochée** ou **false sinon**. Le texte associé à la case à cocher qui identifie son rôle est la valeur de la propriété **Text**.

Exemple:

À partir de 3 symptômes identifiés, prédire le diagnostic correspondant selon les indications suivantes.

Symptômes: Poumon infecté: Oui ou Non
 Maux de Gorge: Oui ou Non
 Température: Oui ou Non

Le patient a le poumon infecté et fait de la température: le diagnostic est **Pneumonie**.

Le patient a des maux de gorge et fait de la température: le diagnostic est **Amygdalite**.

Le patient a des maux de gorge mais ne fait pas de température: le diagnostic est **Rhume**.

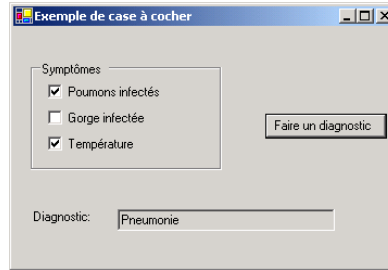
Le patient fait seulement de la température: le diagnostic est **Infection virale**.

Le patient n'a aucun symptôme: le diagnostic est **Parfaite santé**.

Attention: Dans certains cas, il n'y a aucun diagnostic et dans d'autres cas, il y a 2 diagnostics.

The figure displays four screenshots of a graphical user interface window titled "Exemple de case à cocher". Each window contains a group box labeled "Symptômes" with three checkboxes: "Poumons infectés", "Gorge infectée", and "Température". To the right of the checkboxes is a button labeled "Faire un diagnostic". Below the checkboxes is a text field labeled "Diagnostic:".

- Top-left screenshot:** All three checkboxes are unchecked. The "Diagnostic:" field contains the text "Parfaite Santé".
- Top-right screenshot:** All three checkboxes are checked. The "Diagnostic:" field contains the text "Pneumonie Amygdalite".
- Bottom-left screenshot:** The "Gorge infectée" checkbox is checked, while "Poumons infectés" and "Température" are unchecked. The "Diagnostic:" field contains the text "Rhume".
- Bottom-right screenshot:** The "Poumons infectés" checkbox is checked, while "Gorge infectée" and "Température" are unchecked. The "Diagnostic:" field contains the text "Aucun diagnostic possible".



Algorithme du bouton Faire un diagnostic:

```

Lis PoumonInfecte, Temperature, MalGorge
Diagnostic ← ""
Si PoumonInfecte = VRAI ET Temperature = VRAI
    Diagnostic ← "Pneumonie"
Si MalGorge = VRAI
    Si Temperature = VRAI
        Diagnostic ← Diagnostic + "Amygdalite"
    Sinon
        Diagnostic ← Diagnostic + "Rhume"
Si MalGorge = FAUX ET PoumonInfecte = FAUX
    Si Temperature = VRAI
        Diagnostic ← Diagnostic + "Infection virale"
    Sinon
        Diagnostic ← Diagnostic + "Parfaite santé"
Si Diagnostic = ""
    Diagnostic ← "Aucun diagnostic possible"
Écris Diagnostic
  
```

Traduction de l'algorithme du bouton en C#:

```

private void btnDiagnostic_Click(object sender, System.EventArgs e)
{
    string Diagnostic = "";

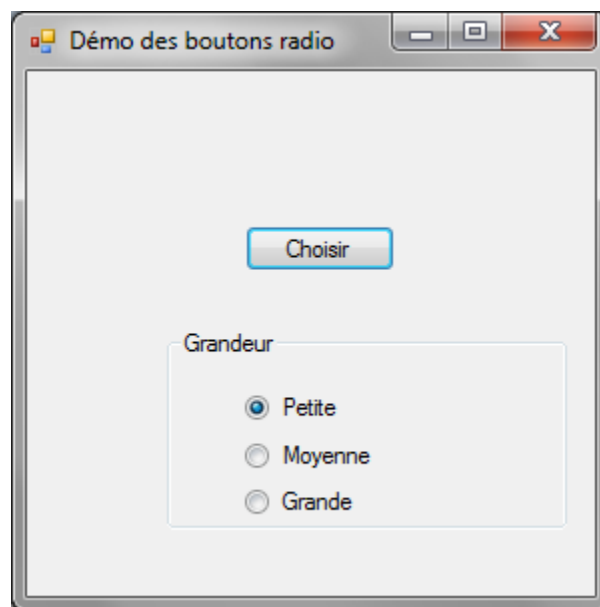
    if (cbPoumons.Checked && cbTemperature.Checked)
        Diagnostic = "Pneumonie";
    if (cbGorge.Checked)
        if (cbTemperature.Checked)
            Diagnostic = Diagnostic + " Amygdalite";
        else
            Diagnostic = Diagnostic + " Rhume";
    if (cbGorge.Checked == false && cbPoumons.Checked == false)
        if (cbTemperature.Checked)
            Diagnostic = Diagnostic + " Infection virale";
        else
            Diagnostic = "Parfaite Santé";
    if (Diagnostic == "")
        Diagnostic = "Aucun diagnostic possible";

    txtDiagnostic.Text = Diagnostic;
}
  
```

Note: L'opérateur + utilisé sur une variable chaîne s'appelle l'opérateur de concaténation et permet mettre bout à bout deux chaînes.

8.7 L'utilisation des contrôles Boutons Radio

Les boutons radio sont des contrôles que l'on peut utiliser sur un formulaire lorsque l'utilisateur de l'application doit choisir une option parmi plusieurs. Les boutons radio sont placés à l'intérieur d'un groupe (GroupBox) et chaque groupe est indépendant. À l'intérieur d'un groupe, seulement un bouton radio peut être sélectionné. La propriété **Checked** est de **type booléen** et vaut **true si l'option est sélectionnée** ou **false sinon**. Le texte associé au bouton radio qui identifie son rôle est la valeur de la propriété **Text**. Pour obliger l'utilisateur à faire un choix, on peut initialiser à la conception la propriété **Checked** d'un des boutons radio de chaque groupe à **true**.



La propriété **Tag** de chaque bouton radio peut contenir une valeur. Cette valeur servira dans le programme. On affecte la bonne valeur soit par la page des propriétés du bouton radio soit par le constructeur du formulaire. La deuxième méthode est à privilégier puisque d'éventuelles modifications des valeurs se feront dans le code.

Une seule méthode est associée à l'événement **CheckedChanged** pour tous les boutons radio du groupe. Sélectionner tous les boutons radio du groupe et nommer la méthode associée à l'événement **CheckChanged**.

Dans l'algorithme nous gardons le **lis** et la validation donc rien ne change.

En C# :

Nous avons besoin d'une variable membre `m_Grandeur` que l'on peut initialiser dans le constructeur au choix par défaut pour ce groupe de boutons radio.

Nous associons, à l'ensemble des boutons radio du groupe, une seule méthode associée à l'évènement `CheckedChanged` :

```
private void rbGrandeur_CheckedChanged(object sender, EventArgs e)
{
    RadioButton Choix = (RadioButton)sender;
    m_Grandeur = Choix.Tag.ToString();
}
```

Plusieurs méthodes statiques de conversion sont disponibles dans la classe `Convert`.

La propriété `Tag` du contrôle n'a pas de type. La façon la plus simple est de la convertir en chaîne pour ensuite utiliser une méthode de conversion selon l'utilisation que le programme doit en faire.

8.8 L'utilisation des ComboBox

Un contrôle de type **ComboBox** est un contrôle qui combine une boîte de saisie et une liste. L'utilisateur peut soit taper du texte dans la boîte de saisie, soit sélectionner un élément de la liste. Le ComboBox dispose d'une **propriété Text** comme celle des boîte de saisie (TextBox) où l'on peut aussi bien afficher du texte que saisir les données de l'utilisateur. La **propriété Items** contient les chaînes apparaissant dans la liste et la **propriété SelectedIndex** contient le numéro de l'élément sélectionné (**le premier = position 0**).

La propriété **DropDownStyle** détermine le mode de fonctionnement de la liste déroulante. Par défaut sa valeur est **DropDown**, les autres valeurs possibles sont : **Simple** et **DropDownList**.

DropDown : L'utilisateur peut saisir du texte dans la boîte de saisie ou dérouler la liste (en cliquant sur la flèche) pour choisir un élément.

Simple : L'utilisateur peut modifier le contenu de la boîte de saisie en inscrivant une nouvelle valeur, mais il ne peut dérouler la liste. (équivalent au TextBox)

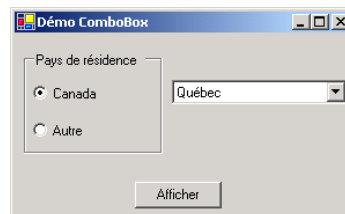
DropDownList : L'utilisateur ne peut inscrire directement du texte dans la boîte de saisie. Il doit dérouler la liste pour en sélectionner un élément.

L'utilisation d'un ComboBox facilite la saisie pour l'utilisateur tout en éliminant la validation nécessaire en spécifiant une valeur initiale à la propriété Text ou SelectedIndex.

L'exemple qui suit utilise un groupe de boutons radio pour sélectionner le pays de résidence (Canada ou Autre) et utilise un ComboBox pour saisir la province ou l'état.

Si l'utilisateur sélectionne le Canada, il pourra dérouler la liste des provinces pour en sélectionner une, sinon il devra inscrire le nom de la province dans la boîte de saisie. C'est dans la propriété Items du ComboBox qu'on inscrit la liste des valeurs.

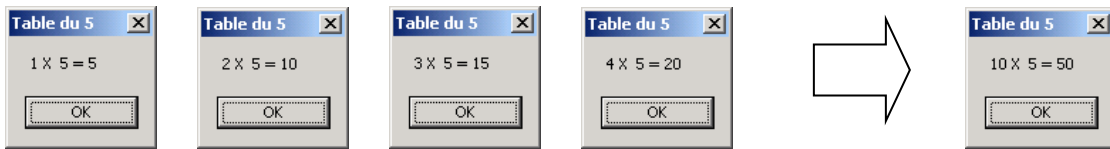
Le bouton radio Canada est sélectionné au départ et le ComboBox est initialisé à Québec de style DropDownList. Un gestionnaire d'événement est déclenché lorsque l'on clique sur le bouton radio Canada pour réinitialiser le style et la valeur par défaut du ComboBox.



8.9 L'affichage dans une boucle

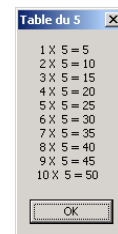
8.9.1 Affichage dans une boucle en utilisant plusieurs MessageBox

```
private void btnTextBox_Click(object sender, System.EventArgs e)
{
    const int Multiplicateur = 5;
    string Affichage;
    int Multiplicande = 1, Resultat;
    while (Multiplicande <= 10)
    {
        Resultat = Multiplicande * Multiplicateur;
        Affichage = Multiplicande.ToString() + " X "
            + Multiplicateur.ToString() + " = "
            + Resultat.ToString();
        MessageBox.Show(Affichage, "Table du 5");
        Multiplicande++;
    }
}
```



8.9.2 L'affichage dans une boucle en utilisant un seul MessageBox

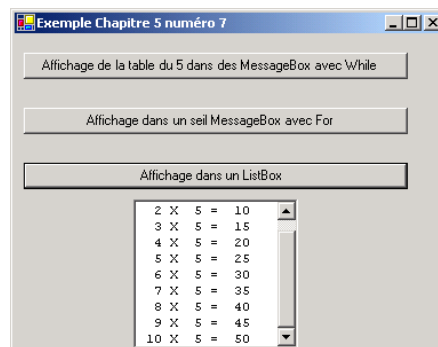
```
private void btn1SeulMessageBox_Click(object sender,
System.EventArgs e)
{
    const int Multiplicateur = 5;
    string Ligne, AffichageComplet="";
    int Multiplicande, Resultat;
    for (Multiplicande = 1; Multiplicande <= 10; Multiplicande++)
    {
        Resultat = Multiplicande * Multiplicateur;
        Ligne = Multiplicande.ToString() + " X "
            + Multiplicateur.ToString() + " = "
            + Resultat.ToString();
        AffichageComplet = AffichageComplet + Ligne;
    }
    MessageBox.Show(AffichageComplet, "Table du 5");
}
```



8.9.3 L'affichage dans une boucle en utilisant un ListBox

Le contrôle **ListBox** permet d'afficher une liste d'éléments dans laquelle l'utilisateur peut sélectionner en cliquant. Contrairement à un ComboBox, un ListBox ne possède pas de propriété Text permettant à l'utilisateur d'entrer un nouvel élément dans la liste. La propriété Item contient les éléments de la liste. La propriété SelectedItem contient l'élément actuellement sélectionné et la propriété SelectedIndex contient l'indice de l'élément actuellement sélectionné. La méthode Add permet d'ajouter un nouvel élément dans la liste.

```
private void button3_Click(object sender, System.EventArgs e)
{
    const int Multiplicateur = 5;
    string Ligne;
    int Multiplicande, Resultat;
    for (Multiplicande = 1; Multiplicande <= 10; Multiplicande++)
    {
        Resultat = Multiplicande * Multiplicateur;
        Ligne = Multiplicande.ToString() + " X "
                + Multiplicateur.ToString() + " = "
                + Resultat.ToString();
        lbTable.Items.Add(Ligne);
    }
}
```



Exemple d'utilisation des propriétés d'un ListBox

```
private void lbTable_SelectedIndexChanged(object sender, System.EventArgs e)
{
    MessageBox.Show(lbTable.SelectedItem.ToString());
    MessageBox.Show(lbTable.SelectedIndex.ToString());
}
```

8.10 La classe string

8.10.1 Définition

Le type **string** est un alias de **System.String** c'est-à-dire la classe qui représente une chaîne de caractères ; autrement dit, une série de caractères Unicode.

8.10.2 Les opérateurs d'égalités

Les opérateurs d'égalité (== et !=) sont définis afin de comparer les valeurs des objets string. Le test d'égalité des chaînes est ainsi plus intuitif.

```
string A = "DFGC";  
string B = "AABB"  
string C = "DFGC";
```

```
if (A == B)  
if (A == C)  
if (A!=C)
```

8.10.3 L'opérateur de concaténation (+)

L'opérateur + est redéfini pour agir sur 2 chaînes.

```
string Nom = "Asselin";  
string Prenom = "Christian";  
string NomComplet = Nom + " , " + Prenom
```

8.10.4 La propriété Length

La propriété Length est utilisée pour obtenir le nombre de caractères dans une chaîne de caractère de type string.

```
public int Length {get;} // le get indique en lecture seulement
```

Valeur de propriété

Nombre de caractères dans cette instance.

Notes

La propriété Length retourne le nombre d'objets [Char](#) dans cette instance, et non le nombre de caractères Unicode. Cela s'explique par le fait qu'un caractère Unicode peut être représenté par plusieurs Char.

Exemple:

```
string Nom;  
int Longueur;  
Nom = "Asselin";  
Longueur = Nom.Length;  
MessageBox.Show(Longueur.ToString());
```

Affichera: 7

8.10.5 Accéder à un caractère d'une chaîne

L'indexeur de la classe String permet d'obtenir le caractère à une position de caractère spécifiée dans une chaîne de caractères de type string.

```
public char this [int] {get;} // le get indique en lecture seulement
```

Paramètres

index : Position du caractère dans cette instance.

Valeur de propriété

Caractère Unicode.

Notes

Le paramètre index est de base zéro.

Cette propriété retourne le [Char](#) à la position spécifiée par le paramètre index. Toutefois, un caractère Unicode peut être représenté par plusieurs Char. Il n'y a pas de propriété en écriture permettant d'écrire un caractère à une position donnée dans la chaîne. Voir 9.6.6.2.

Exemple:

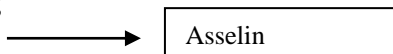
```
if (Nom[2] == 'a')
    MessageBox.Show("OK");
```

8.10.6 Les méthodes de base de la classe string

8.10.6.1 La méthode string Insert(Position, ChaîneÀInsérer)

Cette méthode insère une chaîne de caractères de type string au point d'index indiqué dans une chaîne de caractères de type string et **retourne la nouvelle chaîne**.


```
string Nom = "Aslin";
Nom = Nom.Insert(2,"se");
MessageBox.Show(Nom);
```



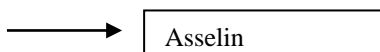
8.10.6.2 La méthode string Remove(Position , LongueurÀEnlever)

Cette méthode supprime un nombre spécifié de caractères d'une chaîne de caractères de type string, en commençant à une position définie et **retourne la nouvelle chaîne**.

```
string Nom = "Asselin";
Nom = Nom.Remove(1,1);
MessageBox.Show(Nom);
```



```
string Nom = "Asvelin";
Nom = Nom.Remove(2,1);
Nom = Nom.Insert(2,"s");
MessageBox.Show(Nom);
```



8.10.6.3 La méthode string Replace()

Format: string Replace(ChaineÀRemplacer, ChaineDeRemplacement)

Remplace toutes les occurrences d'un caractère Unicode spécifié ou [String](#) dans cette instance par un autre caractère Unicode spécifié ou **String** et **retourne la nouvelle chaîne**.

```
string errString = "Ce docment utilise 3 autres docments pour documenter la  
documentation.";  
string CorrectString;  
CorrectString = errString.Replace("docment", "document");  
MessageBox.Show(CorrectString);
```

—————→ Ce document utilise 3 autres documents pour documenter la documentation.

8.10.6.4 La méthode string ToLower()

Retourne une copie de [String](#) en minuscules.

```
string Nom = "Asselin, Christian";  
Nom = Nom.ToLower();  
MessageBox.Show(Nom);
```

asselin, christian

8.10.6.5 La méthode string ToUpper()

Retourne une copie de [String](#) en majuscules.

```
string Nom = "Asselin, Christian";  
Nom = Nom.ToUpper();  
MessageBox.Show(Nom);
```

ASSELIN, CHRISTIAN

8.10.7 Les autres méthodes de la classe string

Il existe un grand nombre de méthodes de la classe String. En effet, la manipulation de chaînes de caractères est très fréquente en programmation. Consultez l'aide sous la rubrique String / méthodes pour trouver la méthode que vous recherchez.

8.11 La structure char

Représente un caractère Unicode.

Quelques méthodes statiques qui manipulent un objet de type char

Nom	Description
IsDigit	Indique si un caractère Unicode est classé dans la catégorie des chiffres décimaux.
IsLetter	Indique si un caractère Unicode est classé dans la catégorie des caractères alphabétiques.
IsLetterOrDigit	Indique si un caractère Unicode est classé dans la catégorie des caractères alphabétiques ou des chiffres décimaux.
IsLower	Indique si un caractère Unicode est classé dans la catégorie des lettres minuscules.
IsNumber	Indique si un caractère Unicode est classé dans la catégorie des nombres.
IsPunctuation	Indique si un caractère Unicode est classé dans la catégorie des signes de ponctuation.
IsUpper	Indique si un caractère Unicode est classé dans la catégorie des lettres majuscules.
IsWhiteSpace	Indique si un caractère Unicode est classé dans la catégorie des espaces blancs.
ToLower	Convertit la valeur d'un caractère Unicode en son équivalent en minuscules.
ToString	Convertit la valeur de cette instance en sa représentation sous forme de chaîne équivalente.
ToUpper	Convertit la valeur d'un caractère Unicode en son équivalent en majuscule.

8.11.1 Des exemples d'utilisation

```
char aa = 'a';
char bb;
bool retour;

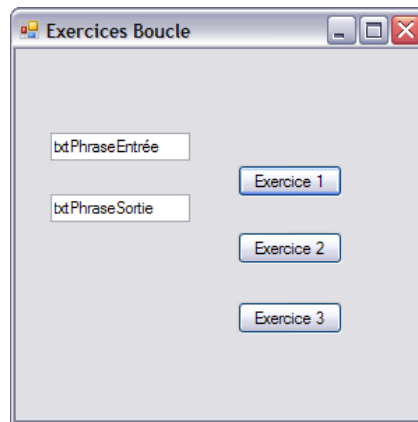
bb = char.ToUpper(aa); // bb vaut 'A'
retour = Char.IsPunctuation(aa)); // retour vaut faux
```

Cette méthode transforme une valeur saisie dans un TextBox en une variable de type char, transforme la variable en majuscule; Vérifie si le caractère est une lettre majuscule, si oui affiche un message. Affiche ensuite le caractère en majuscule.

```
private void btnTest_Click(object sender, EventArgs e)
{
    char Var1;
    bool Resultat;
    Resultat = Char.TryParse(txtCode.Text, out Var1);
    if(Resultat == false)
    {
        MessageBox.Show("Donnée invalide");
        return;
    }
    Var1 = char.ToUpper(Var1);
    if (char.IsLetter(Var1))
        MessageBox.Show("C'est une lettre");
    MessageBox.Show(Var1.ToString());
}
```

8.12 Des exercices

Soit la formulaire suivante:

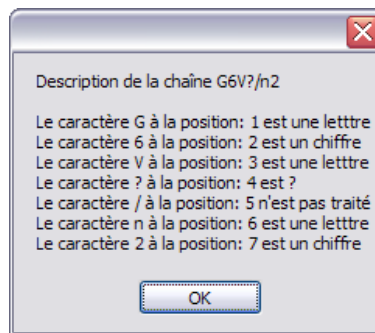


Exercice 1. Quel sera le contenu de la boîte d'édition à l'exécution de la méthode suivante. Le texte entré dans la boîte d'édition txtPhraseEntrée est : ABRACADABRA

```
private void btnEx1_Click(object sender, EventArgs e)
{
    string Chaine1 = txtPhraseEntrée.Text;
    string Chaine2 = "";
    int i;
    for (i = Chaine1.Length - 1; i >= 0; i--)
    {
        Chaine2 = Chaine2 + Chaine1[i].ToString();
    }
    txtPhraseSortie.Text = Chaine2;
}
```

Exercice 2. Qu'est-ce qui sera affiché à l'exécution de la méthode suivante? La valeur saisie dans la boîte d'édition txtPhraseEntrée est : G6V?/n2

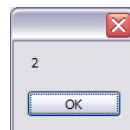
```
private void Ex2_Click(object sender, EventArgs e)
{
    string Var1, cs = "", Affiche ;
    int Pos;
    Var1 = txtPhraseEntrée.Text;
    Affiche = "Description de la chaîne " + Var1 + "\n";
    for (Pos = 0; Pos < Var1.Length; Pos = Pos + 1)
    {
        if (char.IsLetter(Var1[Pos]) == true)
            cs = "\nLe caractère "+ Var1[Pos]+ " à la position: "+ (Pos+1) + " est une letttrre");
        else
            if (char.IsNumber(Var1[Pos]) )
                cs = "\nLe caractère "+ Var1[Pos]+ " à la position: "+ (Pos+1) + " est un chiffre");
            else
                if (Var1[Pos] == '?')
                    cs = "\nLe caractère "+ Var1[Pos]+ " à la position: "+ (Pos+1) + " est ?");
                else
                    cs = "\nLe caractère "+ Var1[Pos]+ " à la position: "+
                        (Pos+1) + " n'est pas traité");
        Affiche = Affiche + cs;
    }
    MessageBox.Show(Affiche);
}
```



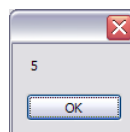
Exercice 3.

```
private void Ex3_Click(object sender, EventArgs e)
{
    string Varx = txtPhraseEntrée.Text;
    int v = 0;
    while (v < Varx.Length && char.IsWhiteSpace(Varx[v]) == false)
        v = v+1;
    MessageBox.Show(v.ToString());
}
```

- a) **Qu'est-ce qui sera affiché à l'exécution de la méthode suivante? La valeur saisie dans la boîte d'édition txtPhraseEntrée est : Le troisième jour.**



- b) **La valeur saisie dans la boîte d'édition txtPhraseEntrée est : Monde**



8.13 Les raccourcis d'expression arithmétiques et les méthodes mathématiques

8.13.1 Les opérateurs équivalents en C#

L'opérateur `++` peut être utilisé pour incrémenter une variable.

Exemple: `nNombre = nNombre + 1;` peut être remplacé par: `nNombre++;`



int A = 3, B;

B = A++;
// B vaut 3 et A vaut 4

Le résultat de l'opération représente la valeur de l'opérande avant son incrémentation.

B = ++A;
// B vaut 4 et A vaut 4

Le résultat de l'opération représente la valeur de l'opérande après son incrémentation.

L'opérateur `--` peut être utilisé pour décrémenter une variable.

Exemple: `nNombre = nNombre - 1;` peut être remplacé par: `nNombre--;`

L'opérateur `+=` peut être utilisé pour additionner une valeur à une variable.

Exemple: `dTotal = dTotal + dPrix;` peut être remplacé par: `dTotal += dPrix;`

L'opérateur `-=` peut être utilisé pour soustraire une valeur d'une variable.

Exemple: `dTotal = dTotal - dPrix;` peut être remplacé par: `dTotal -= dPrix;`

8.13.2 La classe Math

Cette classe fournit des constantes et des méthodes de classe (**static**) pour des méthodes trigonométriques, logarithmiques et d'autres méthodes mathématiques courantes.

Champs publics

E Représente la base de logarithme naturelle spécifiée par la constante **e**.

PI Représente le rapport de la circonférence d'un cercle à son diamètre, spécifié par la constante **π**.

Méthodes publiques

Abs Surchargé. Retourne la valeur absolue d'un nombre spécifié.

Acos Retourne l'angle dont le cosinus est le nombre spécifié.

Asin Retourne l'angle dont le sinus est le nombre spécifié.

Atan	Retourne l'angle dont la tangente est le nombre spécifié.
Atan2	Retourne l'angle dont la tangente est le quotient de deux nombres spécifiés.
BigMul	Génère le produit intégral de deux nombres 32 bits.
Ceiling	Retourne le plus petit nombre entier supérieur ou égal au nombre spécifié.
Cos	Retourne le cosinus de l'angle spécifié.
Cosh	Retourne le cosinus hyperbolique de l'angle spécifié.
DivRem	Surchargé. Retourne le quotient de deux nombres.
Exp	Retourne e élevé à la puissance spécifiée.
Floor	Retourne le plus grand nombre entier inférieur ou égal au nombre spécifié.
IEERemainder	Retourne le reste de la division d'un nombre spécifié par un autre.
Log	Surchargé. Retourne le logarithme d'un nombre spécifié.
Log10	Retourne le logarithme de base 10 d'un nombre spécifié.
Max	Surchargé. Retourne le plus grand de deux nombres spécifiés.
Min	Surchargé. Retourne le plus petit de deux nombres.
Pow	Retourne un nombre spécifié élevé à la puissance spécifiée.
Round	Surchargé. Retourne le nombre le plus proche de la valeur spécifiée.
Sign	Surchargé. Retourne une valeur indiquant le signe d'un nombre.
Sin	Retourne le sinus de l'angle spécifié.
Sinh	Retourne le sinus hyperbolique de l'angle spécifié.
Sqrt	Retourne la racine carrée d'un nombre spécifié.
Tan	Retourne la tangente de l'angle spécifié.
Tanh	Retourne la tangente hyperbolique de l'angle spécifié.

8.13.3 Des exemples d'utilisation

La valeur absolue d'un entier signé : **int ValeurAbsolue = Math.Abs(-804128);**

La racine carrée d'un entier signé : **int RacineCarree = Math.Sqrt(25);**

9 Quelques notions de graphisme en C#

9.1 Le graphique GDI+

L'interface graphique (GDI, *Graphics Design Interface*) Windows, appelée GDI+, permet de créer des graphiques, de dessiner du texte et de manipuler des images graphiques en tant qu'objets. Cette interface est conçue pour allier performances et simplicité d'utilisation. Vous pouvez l'utiliser en vue du rendu des images graphiques sur des Windows Forms et des contrôles.

Avant de pouvoir dessiner des traits et des formes, de rendre du texte ou d'afficher et de manipuler des images avec GDI+, vous devez créer un objet `System.Drawing.Graphics`. L'objet **Graphics** représente une surface de dessin GDI+ et l'objet utilisé pour créer des images graphiques.

Deux étapes sont nécessaires à l'utilisation de graphiques :

1. Création d'un objet **Graphics**.
2. Utilisation de l'objet **Graphics** pour dessiner des traits et des formes, pour rendre du texte ou pour afficher et manipuler des images.

9.1.1 La création d'un objet Graphics

Un objet `Graphics` peut être créé en appelant la méthode `CreateGraphics` d'un contrôle ou d'un formulaire afin d'obtenir une référence à un objet **Graphics** représentant la surface de dessin de ce contrôle ou formulaire, cette méthode étant utilisée si vous voulez dessiner sur un formulaire ou un contrôle existant. C'est la manière présentée dans ce chapitre.

Pour créer un objet Graphics à l'aide de la méthode CreateGraphics

Appelez la méthode **CreateGraphics** du formulaire ou contrôle sur lequel vous voulez rendre des graphiques.

```
// Déclarer un objet de la classe Graphics
Graphics g;
// Affecter à g l'objet graphique représentant la surface
// de dessin du contrôle dont g est membre
g = CreateGraphics();
```

9.1.2 Le dessin et la manipulation de formes et d'images

Une fois l'objet **Graphics** créé, il peut être employé pour dessiner des traits et des formes, pour rendre du texte ou pour afficher et manipuler des images. Les principaux objets utilisés avec l'objet **Graphics** sont les suivants :

- **Pen, classe** - Est utilisée pour dessiner des traits, tracer le contour de formes ou rendre d'autres représentations géométriques.
- **Brush, classe** - Est utilisée pour remplir des zones de graphiques, telles que du texte, des images ou des formes remplies.
- **Font, classe** - Décrit les formes à utiliser pour le rendu de texte.
- **Color, structure** - Représente les différentes couleurs à afficher.

9.2 Les stylets, brosses et couleurs

Vous utilisez des objets de type brosse et stylet pour rendre des graphiques, du texte et des images avec GDI+.

Un **stylet** est une instance de la classe **Pen** et s'utilise pour dessiner des traits et formes avec contour.

Une **brosse** est une instance de n'importe quelle classe dérivée de la classe **Brush** (**abstraite**) **MustInherit** ; elle peut être utilisée pour remplir des formes ou peindre du texte.

Les objets **Color** sont des instances de classes représentant **une couleur particulière** et peuvent être utilisés par des stylets et des brosses pour indiquer la couleur des graphiques rendus. L'exemple ci-dessous illustre l'utilisation de ces objets :

```
// Création d'un stylet qui dessinera en rouge
Pen Stylet = new Pen(Color.Red);
// Création d'une brosse qui remplira en bleu
SolidBrush Brosse = new SolidBrush(Color.Blue);
```

9.2.1 Les stylets

Un stylet sert à dessiner des traits et des courbes ainsi qu'à tracer le contour de formes. L'exemple suivant montre comment créer un stylet noir rudimentaire :

```
// Création d'un stylet qui dessinera en noir
// avec l'épaisseur de 1 par défaut
Pen Stylet1 = new Pen(Color.Black);
// Création d'un stylet qui dessinera en noir
// avec une épaisseur de 5
Pen Stylet2 = new Pen(Color.Black, 5);
```

Vous pouvez également créer un stylet à partir d'un objet brosse existant. Le code ci-dessous illustre la création d'un stylet basé sur une brosse existante, appelée myBrush.


```
// Création d'une brosse rouge
SolidBrush BrosseRouge = new SolidBrush(Color.Red);
// Création d'un stylet qui dessinera en rouge
// avec l'épaisseur de 1 par défaut
Pen Stylet1 = new Pen(BrosseRouge);
// Création d'un stylet qui dessinera en rouge
// avec une épaisseur de 5
Pen Stylet2 = new Pen(BrosseRouge, 5);
```

Après avoir créé un stylet, vous pouvez l'utiliser pour dessiner un trait, un arc ou une forme avec contour. L'exemple ci-dessous illustre l'emploi d'une brosse pour dessiner une ellipse :

```
Pen Stylet = new Pen(Color.Black);
Graphics g = CreateGraphics();
g.DrawEllipse(Stylet, 20, 30, 10, 50);
```

Le stylet créé, vous pouvez modifier plusieurs propriétés régissant l'apparence des traits qu'il trace. Des propriétés telles que **Width** et **Color** influent sur l'apparence du trait, alors que les propriétés **StartCap** et **EndCap** vous permettent d'ajouter des formes personnalisées ou prédéfinies au début ou à la fin du trait. La propriété **DashStyle** vous permet de sélectionner le style de trait, c'est-à-dire notamment plein, en pointillés, en tirets ou en tirets personnalisés, alors que la propriété **DashCap** s'emploie pour personnaliser les extrémités des tirets composant les traits.

9.2.2 Les brosses

Les brosses sont des objets qui, associés à un objet **Graphics**, permettent de créer des formes pleines et de rendre du texte. Il existe plusieurs types de brosses :

Classe Brush	Description
SolidBrush	Forme la plus simple d'une brosse, qui peint dans une couleur unie.
HatchBrush	Est semblable à une brosse SolidBrush , mais permet de choisir un des nombreux modèles prédéfinis au lieu d'utiliser une couleur unie.
TextureBrush	Peint à l'aide d'une texture, comme une image.
LinearGradientBrush	Peint avec un dégradé de deux couleurs.
PathGradientBrush	Peint avec un dégradé complexe de couleurs qui se mélangent en suivant un tracé unique défini par le développeur.

Toutes ces classes sont héritées de la classe **Brush**, qui est une classe (**MustInherit**) abstraite ne pouvant pas être instanciée.

Couleur unie

L'exemple suivant montre comment dessiner une ellipse de couleur rouge unie sur un formulaire. Cette ellipse s'inscrit dans la taille du rectangle qui lui est fourni (dans ce cas, il s'agit du **ClientRectangle** représentant le formulaire tout entier).

```
Graphics g = CreateGraphics();
```

```
SolidBrush BrosseRouge = new SolidBrush(Color.Red);  
g.FillEllipse(BrosseRouge, ClientRectangle);
```

9.2.3 Les couleurs

La structure **Color** du .NET Framework est utilisée pour représenter différentes couleurs. Les couleurs sont employées avec des stylets et des brosses pour déterminer la couleur du rendu.

Couleurs définies par le système

Il existe plusieurs couleurs système accessibles par l'intermédiaire de la structure **Color**. En voici quelques exemples :

```
Color UneCouleur;  
UneCouleur = Color.Red;  
UneCouleur = Color.Aquamarine;  
UneCouleur = Color.LightGoldenrodYellow;  
UneCouleur = Color.PapayaWhip;  
UneCouleur = Color.Tomato;
```

Chacune des instructions ci-dessus assigne à une couleur système du nom indiqué à **UneCouleur**.

Couleurs définies par l'utilisateur

Vous pouvez également créer des couleurs définies par l'utilisateur à l'aide de la méthode **Color.FromArgb**. Cette méthode permet de spécifier la vivacité des composants rouge, bleu et vert individuels d'une couleur.

```
Color MaCouleur;  
MaCouleur = Color.FromArgb(23, 56, 78);
```

Cet exemple génère une couleur définie par l'utilisateur proche du bleu-gris. Chaque nombre doit être un entier compris entre 0 et 255, où 0 indique l'absence de la couleur en question et 255 indique la saturation complète de cette même couleur. Ainsi, **Color.FromArgb(0,0,0)** rend la couleur noire, et **Color.FromArgb(255,255,255)** rend le blanc.

Contrôle alpha (transparence)

Cette méthode vous permet également de spécifier un composant *alpha*. Alpha représente la transparence aux objets situés derrière le graphique rendu. Les couleurs à contrôle alpha peuvent être utiles afin d'obtenir divers effets de nuances et de transparence. Si vous souhaitez spécifier un composant alpha, il doit être le premier argument des quatre arguments passés à la méthode **Color.FromArgb** et doit être un entier compris entre 0 et 255.

```
Color MaCouleur;  
MaCouleur = Color.FromArgb(127, 23, 56, 78);
```

Cet exemple crée une couleur proche du gris-bleu et transparente à 50 %.

Vous pouvez également créer une couleur à contrôle alpha en spécifiant un composant alpha et une couleur définie précédemment.

```
Color MaCouleur;  
MaCouleur = Color.FromArgb(128, Color.Tomato);
```

9.3 Le dessin de traits et de formes avec GDI+

L'objet **Graphics** comprend des méthodes permettant de dessiner divers types de traits et de formes. Des formes simples ou complexes peuvent être rendues dans des couleurs unies ou transparentes, ou à l'aide de textures d'images et de dégradés définis par l'utilisateur. Les traits, les courbes ouvertes et les formes avec contour sont créées à l'aide d'un objet **Pen**. Pour remplir une zone, comme un rectangle ou une courbe fermée, un objet **Brush** est requis.

Pour dessiner un trait ou une forme avec contour

1. Obtenez une référence à l'objet Graphics à l'aide duquel vous allez dessiner.
2. Créez une instance de la classe Pen à utiliser pour dessiner le trait et définissez éventuellement les propriétés requises.
3. Appelez la méthode adaptée à la forme à dessiner, en fournissant les éventuels paramètres requis. Le tableau ci-dessous répertorie quelques-unes des méthodes les plus courantes. Pour obtenir une liste exhaustive, consultez Graphics, méthodes.

Méthode	Forme
Graphics.DrawLine	Trait. Nécessite des coordonnées indiquant les points de départ et de fin.
Graphics.DrawPolygon	Formes complexes. Peut nécessiter un tableau de coordonnées.
Graphics.DrawRectangl	Rectangle. Nécessite un ou plusieurs objets (par exemple, un objet Rectangle) en tant que paramètres.

```
Graphics g = Button1.CreateGraphics();  
Pen Stylet = new Pen(Color.Red);  
Stylet.Width = 5;  
g.DrawLine(Stylet, 1, 1, 45, 65);
```

Pour dessiner une forme remplie

1. Obtenez une référence à l'objet Graphics à l'aide duquel vous allez dessiner.
2. Créez une instance de la brosse (**Brush**) à l'aide de laquelle vous allez peindre la forme.
3. Appelez la méthode adaptée à la forme à peindre, en fournissant les éventuels paramètres requis. Pour certaines méthodes, comme **FillPolygon**, vous devrez fournir un tableau de points décrivant le contour de la forme à peindre. D'autres méthodes, comme **FillRectangle** ou

FillPath, exigent un objet décrivant la zone à remplir. Pour plus d'informations sur les paramètres requis, consultez la rubrique d'aide de la méthode utilisée. Voici quelques exemples :

```
Graphics g = Button1.CreateGraphics;  
SolidBrush BrosseRouge = new SolidBrush(Color.Red);  
g.FillRectangle(BrosseRouge, new RectangleF(50, 50, 100, 100));  
g.FillPie(BrosseRouge, new Rectangle(110, 110, 300, 300), 0, 90);
```

9.4 Le dessin de texte avec GDI+

Vous pouvez utiliser n'importe quel objet **Graphics** comme surface de rendu de texte. Le rendu de texte nécessite un objet **Brush**, qui indique le modèle de remplissage du texte, et un objet **Font**, qui décrit le modèle à remplir. La police peut être n'importe quelle police nommée installée sur le système, et la brosse peut être de n'importe quel type. Par conséquent, il est possible de peindre du texte à l'aide d'une couleur unie, d'un modèle ou encore d'une image.

Pour rendre une chaîne de texte avec GDI+

1. Obtenez une référence à l'objet **Graphics** à l'aide duquel vous allez dessiner.
2. Créez une instance de la brosse (**Brush**) à l'aide de laquelle vous allez peindre le texte.
3. Créez la police dans laquelle le texte devra être affiché.
4. Appelez la méthode **Graphics.DrawString** de l'objet **Graphics** pour rendre le texte. Si vous fournissez un objet **RectangleF**, le texte fera l'objet d'un retour à la ligne dans le rectangle. Sinon, il débutera aux coordonnées de départ fournies.

```
Graphics g = Button1.CreateGraphics();  
SolidBrush BrosseRouge = new SolidBrush(Color.Red);  
Font MaPolice = new Font("Times New Roman", 24);  
g.DrawString("Look at this text!", myFont, myBrush, 10, 10);
```

9.5 Le rendu d'images avec GDI+

Vous pouvez utiliser l'interface GDI+ pour rendre des images existant sous la forme de fichiers dans vos applications. Pour cela, vous devez créer un nouvel objet d'une classe **Image** (comme **Bitmap**), puis créer un objet **Graphics** qui référence la surface de dessin à utiliser et ensuite appeler la méthode **DrawImage** de l'objet **Graphics**. L'image est peinte sur la surface de dessin représentée par l'objet graphique. Vous pouvez avoir recours à l'**éditeur d'images** pour créer et modifier les fichiers image au moment du design et rendre ces images avec GDI+ au moment de l'exécution. Pour plus d'informations, consultez Éditeur d'images.

Pour rendre une image avec GDI+

1. Créez un objet représentant l'image à afficher. Cet objet doit être un membre d'une classe héritant de la classe **Image**, comme **Bitmap** ou **MetaFile**.

2. Créez un objet **Graphics** représentant la surface de dessin à utiliser.
3. Appelez la méthode `Graphics.DrawImage` de l'objet **Graphics** pour rendre l'image. Vous devez spécifier à la fois l'image à dessiner et les coordonnées indiquant sa position.

```
// Utilisation de System.Environment.GetFolderPath pour obtenir  
// le chemin d'accès au dossier MesImages de l'utilisateur courant
```

```
Bitmap MonBitmap = new Bitmap  
    (System.Environment.GetFolderPath  
        (System.Environment.SpecialFolder.MyPictures));
```

```
//Création de l'objet Graphics représentant la surface de dessin  
// du bouton Bouton1
```

```
Graphics g = Bouton1.CreateGraphics();
```

```
g.DrawImage(MonBitmap, 1, 1);
```

10 Exercices synthèses

10.1 Les Si simples, imbriqués et multiples

Écrire l'algorithme qui lira 3 nombres entiers positifs (à valider) et qui placera la plus grande valeur dans la variable A; les 2 autres dans B et C. Afficher le qualificatif du triangle ainsi formé (plus d'un qualificatifs dans certains cas), sachant que:

- $A \geq B+C$: Pas un triangle
- $A^2 = B^2 + C^2$: Triangle Rectangle
- $A^2 > B^2 + C^2$: Triangle Obtus
- $A^2 < B^2 + C^2$: Triangle Aigu
- 3 côtés égaux : Équilatéral
- 2 côtés égaux : Isocèle
- 3 côtés différents: Scalène

10.2 La validation de chaînes de caractères

Écrire l'algorithme pour lire et valider un numéro de siège de théâtre. Les numéros acceptés doivent avoir la forme suivante:

sr-nn	où	s	représente la section (P, M ou E)
		r	représente la rangée (1, 2, 3 ou 4)
		nn	représente un numéro (de 00 à 29)

De plus, votre algorithme doit afficher la section et la rangée correspondante.

Exemples:

P1-2	Erreur, le numéro de siège est incomplet	
A2-21	Erreur, le premier caractère doit être P, M ou E	
M6-11	Erreur, le deuxième caractère doit être de 1 à 4	
M2+11	Erreur, le troisième caractère doit être '-'	
M2-39	Erreur, le quatrième caractère doit être 0, 1 ou 2	
M2-2A	Erreur, le cinquième caractère doit être entre 0 et 9	
P3-05	Section: P	Rangée: 3
E1-29	Section: E	Rangée: 1

10.3 Les boucles simples

Écrire l'algorithme pour résoudre les problèmes suivants :

Je pars en voyage à Las Vegas pour jouer au casino. En joueur prévoyant, je décide d'un budget global maximum pour jouer. Je décide de jouer un certain montant la première journée et de doubler ce montant à chaque jour. Afficher le nombre de jours que je pourrai jouer ainsi et le montant qu'il me restera en poche. Naturellement, je considère que je ne gagne jamais.

Même énoncé, mais je veux savoir le montant joué et le montant qu'il me reste pour chaque journée.

Respecter l'affichage suivant:

Jour	Montant Joué	Reste
1	100	900
2	200	700
.....		
.....		

11 Travail préparatoire obligatoire en laboratoire.

But :

Démarrer/terminer une session de travail sur micro-ordinateur.
Faire un premier contact avec l'environnement de développement Visual Studio.
Utiliser les principales fonctions de l'éditeur de texte de Visual Studio.
Développer un premier programme de type Application Windows en C#.
Enregistrer un programme sur l'espace réseau.
Récupérer un programme.

Démarrer/terminer une session de travail sur micro-ordinateur.

Pour démarrer une session de travail, il suffit d'ouvrir l'interrupteur du boîtier et de l'écran. Les instructions de démarrage sont stockées dans la mémoire morte de l'ordinateur (ROM) et sont conçues pour chercher le système d'exploitation sur le disque dur de votre ordinateur.

Procédure de LOGIN (2 étapes) :

1ière étape : **Login DA**

Pour **la connexion au réseau**, vous devez entrer comme **nom d'utilisateur votre numéro de DA** (sur votre horaire) et **comme mot de passe votre date de naissance sous le format AAAAMMJJ**.

Ex : **20020823** (pour le 23 aout 2002)

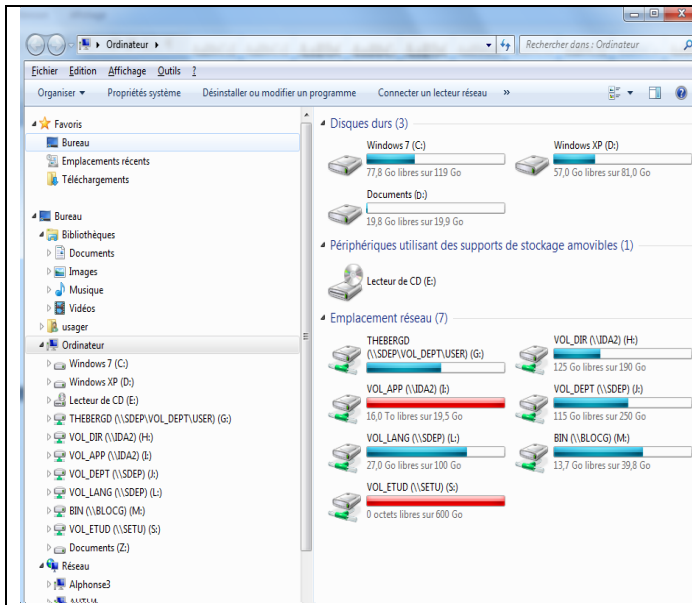
Une fois connecté au réseau, vous disposez de votre espace personnel sur le serveur et de l'accès aux imprimantes du **Bloc G**.

2ième étape : Login sur cet ordinateur (**Login Windows**)

Pour ouvrir **une session locale**, vous devez entrer comme nom d'utilisateur **usager** et comme mot de passe **etudiant** (SANS ACCENT). Pour terminer une session de travail, cliquer sur le bouton Fenêtre Windows



dans le bas gauche de votre écran et cliquer le bouton droit de votre souris pour et choisir arrêt ou se déconnecter pour l'option d'ARRÊTER; ou choisir l'option: Verrouiller ; Redémarrer; Mettre en veille.



Unités locales de disque dur :

(C:) Contient Windows et les logiciels;
(D:) Espace de travail accessible à tous sans sécurité mais accès rapide;
(E:) lecteur de CD;

Unités réseau :

(G:) Votre espace disque personnel accessible de l'extérieur;
(J :) Dépôt de documents accessible aux étudiants en lecture seulement;
(S :) Dépôt de travaux des étudiants accessible en écriture seulement;

Introduction à Visual C#

Visual C# est un langage de programmation de haut niveau orienté objet.

On appelle « langage de programmation » un langage destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter. Un langage de programmation est ainsi une façon pratique pour nous (humains) de donner des instructions à un ordinateur. Les ordinateurs entre eux utilisent plutôt un « protocole de communication ».

À CHAQUE instruction correspond UNE action du processeur.

Le langage C# a été conçu par Microsoft pour sa nouvelle plateforme .Net.

C'est un descendant de C++ avec des caractéristiques de Java et plusieurs autres langages. C# n'est pas le seul langage utilisable sur .Net.

La plate-forme Microsoft.Net

La plateforme Microsoft .Net fournit l'ensemble des outils et technologies nécessaires à la création d'applications Windows et Web à l'aide de plusieurs langages.

L'environnement de développement Visual Studio .Net

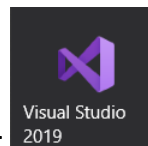
Un premier contact avec l'environnement de développement Microsoft Visual Studio.Net

Pour lancer Microsoft Visual Studio, cliquez sur le bouton **Démarrer** dans la barre des tâches.

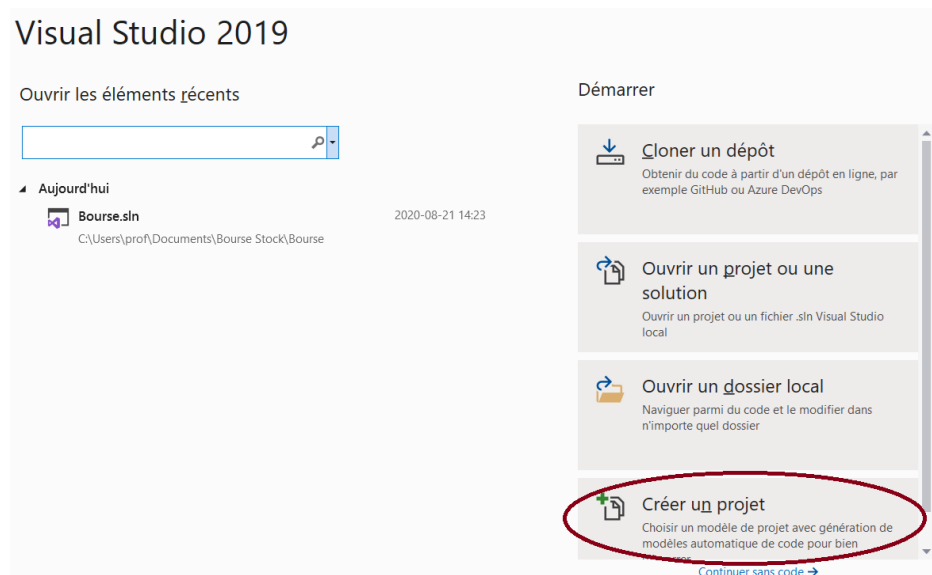
Sélectionnez dans



Visual Studio 2019 l'icon suivant

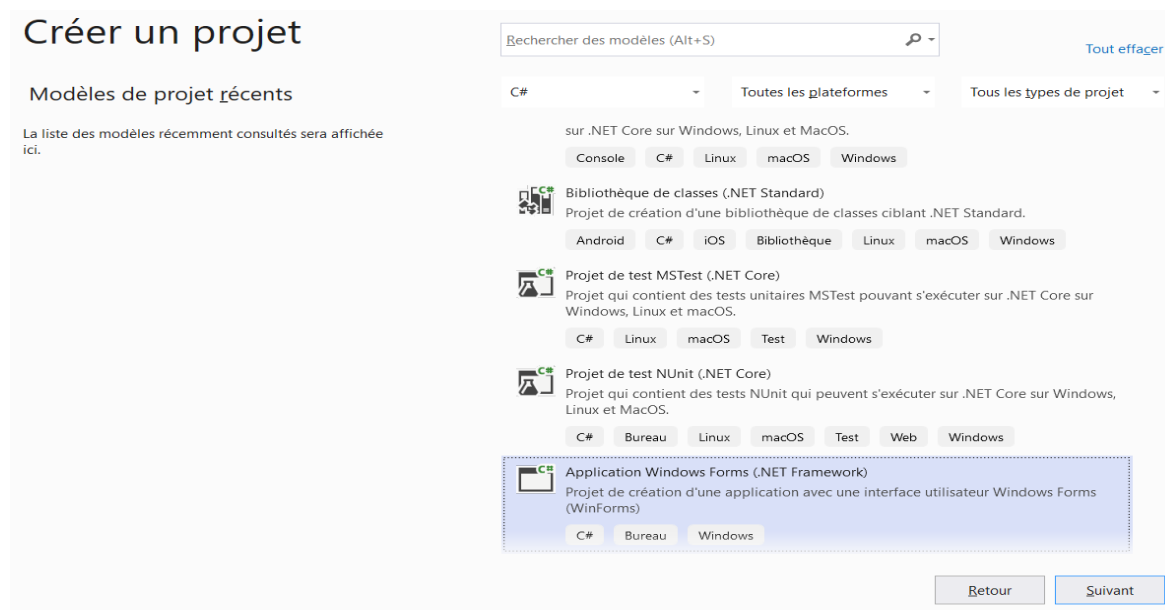


Après quelques instants, la fenêtre de démarrage est affichée.



La fenêtre de démarrage de Microsoft Visual Studio

Choisir Créer un projet



Et par la suite Application Windows Forms(Net FrameWorks) et faire suivant.

Configurer votre nouveau projet

Application Windows Forms (.NET Framework) C# Bureau Windows

Nom du projet

Premier Projet

Emplacement

D:\Asselin

Nom de la solution ⓘ

Premier Projet

☒ Placer la solution et le projet dans le même répertoire

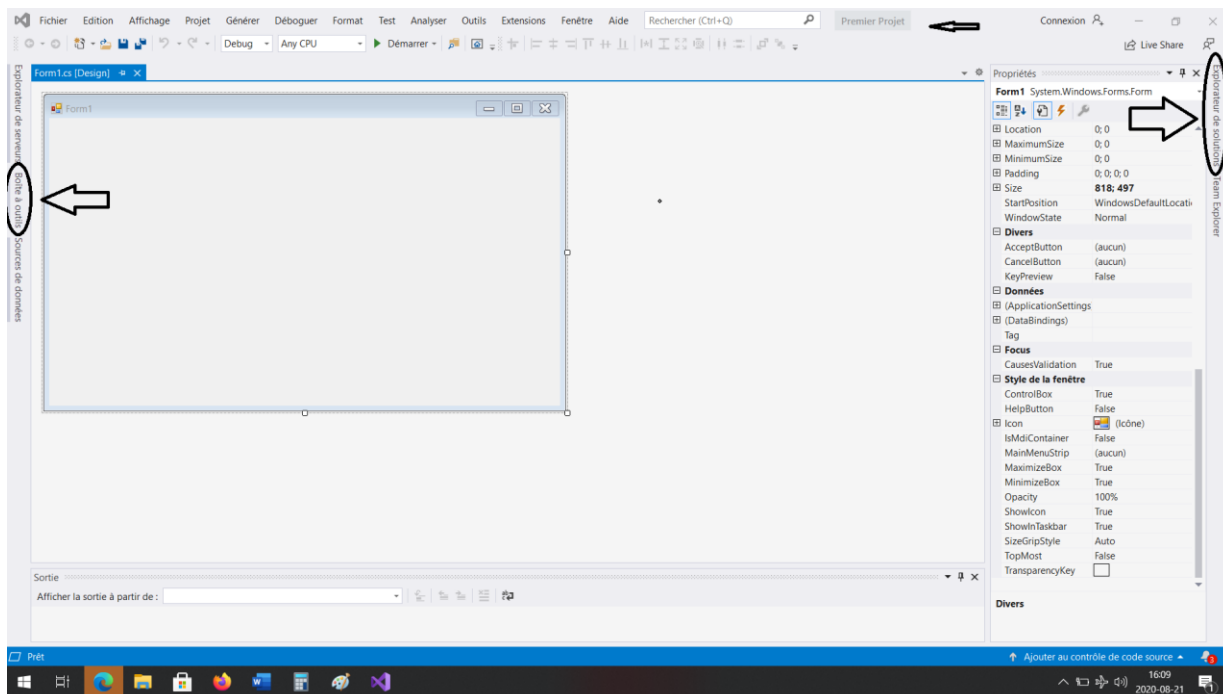
Framework

.NET Framework 4.7.2

Retour

Créer

Cliquer sur le bouton Créer et vous devriez avoir l'écran suivant :



Comme la plupart des applications Windows, Visual Studio possède :

- une barre de menus
- plusieurs barres d'outils
- une barre d'état
- un espace de travail dans lequel seront affichés les programmes et les messages envoyés par le compilateur, l'éditeur de liens et le débogueur

Les commandes de menus de Visual Studio sont accessibles de deux manières différentes.

- **Avec la souris** : pointez Créer un projet sous le titre Nouveau Projet puis cliquez.
- **Avec le raccourci clavier** : Une combinaison de touches spéciales a été affectée aux commandes de menu les plus utilisées. Par exemple Ctrl+Maj+N pour la commande Projet du sous-menu Nouveau du menu Fichier. Ces raccourcis clavier apparaissent en regard des commandes de menu. Avec le temps, retenir ceux qui correspondent aux commandes les plus utilisées. Vous gagnerez ainsi un temps considérable.

La conception d'un programme en C#

Vous allez maintenant écrire votre premier programme Visual C# en mode fenêtre avec le logiciel Visual Studio.Net 2019 !!! Cela implique deux étapes:

- L'étape de conception visuelle
- L'étape de codage

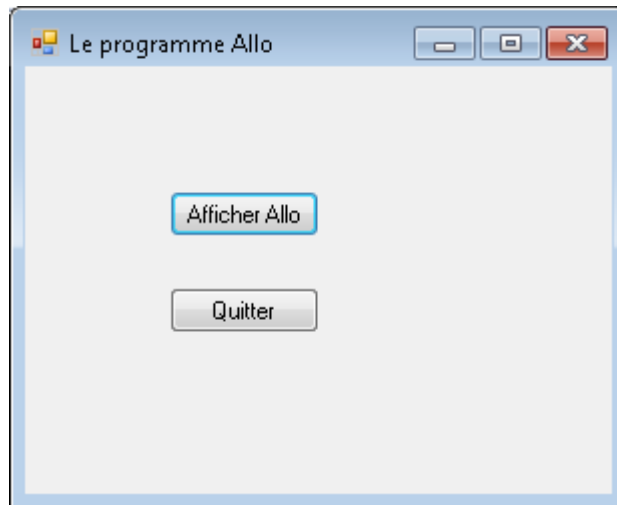
Dans *l'étape de conception visuelle*, vous déterminez l'apparence visuelle de la fenêtre qui servira d'entrée et de sortie. Vous utilisez les outils Visual C# pour placer les différents objets (tels que boutons de commande, barres de défilement, boutons radio, etc.) dans la fenêtre de l'application. Pour la plupart des projets, vous écrivez la logique algorithmique associée à chaque bouton. Pendant l'étape de conception visuelle, vous n'écrivez aucun code.

Dans *l'étape de codage*, vous écrivez le code à l'aide de l'éditeur de texte Visual C# et du langage de programmation C#.

Ce que fera le programme Allo.exe

Avant de commencer l'écriture d'un programme, vous devez tout d'abord spécifier l'aspect qu'aura la fenêtre et ce qu'elle offrira à l'utilisateur, c'est la conception visuelle :

- Le programme Allo.exe comprendra deux boutons de commande: **Afficher Allo** et **Quitter**. Lorsque vous cliquerez sur le bouton Afficher Allo, une boîte de message ALLO apparaîtra;
- Pour fermer la boîte de message ALLO, il faudra cliquer sur son bouton OK;
- Pour terminer le programme Allo.exe, il faudra cliquer sur le bouton de commande Quitter.



Maintenant que vous savez à quoi doit ressembler le programme Allo.EXE et comment il doit réagir, vous pouvez commencer sa conception. Dans les sections suivantes, vous verrez comment créer le programme Hello.exe en suivant pas à pas les instructions.

- ⚠ **Tous vos programmes doivent être développés sur l'unité D: dans un répertoire personnel. Ne développez pas de programmes directement dans votre espace sur le réseau ni directement sur une clé USB au risque de devoir tout recommencer. Lorsque vous avez terminé un programme que vous désirez conserver, copier le dossier dans votre espace réseau ou sur une clé USB.**

Création du projet (À faire, créer un répertoire à votre nom sur le D :)

La première chose à faire lorsque vous créez un nouveau programme C# est de créer le projet du programme. Pour créer le projet du programme Allo, procéder aux étapes suivantes:

- Démarrer **Microsoft Visual Studio 2019**;
- Faire les étapes précédentes ci-haut

→ Spécifier le nom du projet: **Hello** ⚠ il est presque impossible de changer ce nom par la suite

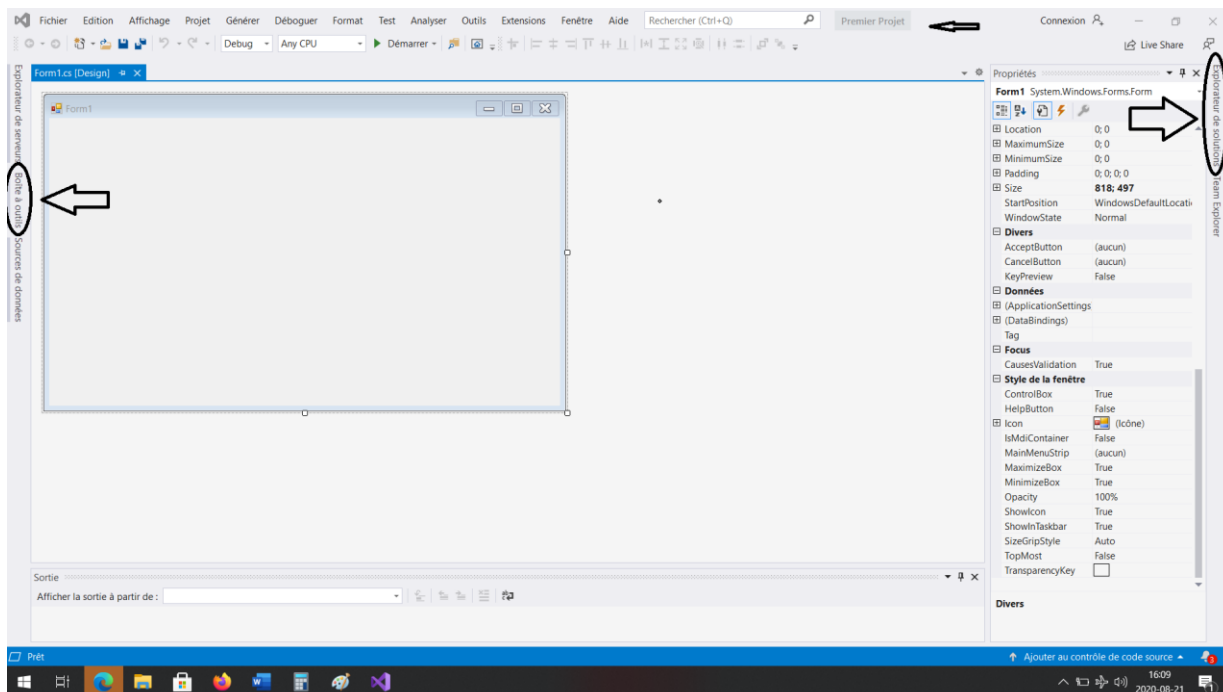
→ Spécifier l'emplacement du projet: **D:\Votre Répertoire** ⚠ obligatoirement sur le D :

Le titre de votre fenêtre Visual Studio est maintenant :

Allo -Microsoft Visual Studio.

Le "Allo" du titre indique que vous travaillez actuellement sur le projet Allo.

Selon le paramétrage du bureau Visual C#, votre bureau Visual C# peut avoir un aspect différent de celui-ci. Identifiez et reprenez les différentes parties de la fenêtre.



Dans la partie gauche de la fenêtre, vous avez un icône qui représente la **boîte à outils** (les différents objets que vous pouvez placer dans votre formulaire. Au centre, votre formulaire en mode conception (onglet **Form1.cs(Design)**). Dans la partie droite, **Explorateur de solutions** et la **fenêtre des propriétés**. ⚠ Si vous n'avez pas ces fenêtres dans votre environnement, faites-les afficher à partir du menu **Affichage**. (Affichage – Explorateur de solutions et Affichage – Fenêtres des propriétés).

Compilation (À faire)

Croyez-le ou non, mais bien que vous n'ayez pas encore écrit une seule ligne de code, vous avez déjà

une application Windows fonctionnelle. Le code trame écrit pour vous par l'assistant fait réellement quelque chose. Pour le constater, vous devez générer le programme Allo avant de l'exécuter.

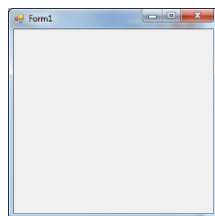
Utilisez le menu **Générer la solution** du menu **Build** pour compiler et lier le programme Allo. (ou encore Générer Allo du menu Build)

Dans l'étape précédente, il vous était demandé de compiler et de lier le programme Allo en sélectionnant Générer Allo dans le menu Build de Visual Studio. Vous pouvez également compiler et lier le programme en sélectionnant Regénérer la solution ou Regénérer Allo dans le menu Build. Lorsque vous choisissez Générer dans le menu Build, seuls les fichiers modifiés depuis la dernière compilation sont compilés à nouveau. Lorsque vous sélectionnez Regénérer, tous les fichiers du programme sont compilés (qu'ils aient ou non été modifiés depuis la dernière compilation). Il peut arriver que votre projet s'égare et que Visual Studio "pense" que des fichiers qui devraient être compilés ne le soient pas. Dans ce cas, vous pouvez forcer Visual Studio à compiler tous les fichiers en sélectionnant Regénérer dans le menu Build.

Exécution (À faire)

Sélectionnez Exécuter sans Débogage dans le menu Déboguer pour exécuter le programme Allo. (Vous pouvez aussi utiliser les touches de raccourci CTRL+F5)

Visual Studio répond en exécutant le programme Allo.exe. La fenêtre principale du programme Allo.exe apparaît.



Comme vous le voyez, le code trame écrit par l'assistant a créé un programme exécutable simple dont la fenêtre contient un formulaire principal vide. La barre de titre contient le titre du formulaire: Form1, ainsi que les 3 icônes de réduction, restauration et fermeture qui sont fonctionnels. Un formulaire est un type de fenêtre sur laquelle on peut déposer différents contrôles de saisie : bouton de commande, boîte de saisie de texte,... C'est un type de fenêtre différent de celui que l'on retrouve, par exemple, sous Bloc Note.

Bien entendu, le programme Allo.exe ne possède encore ni l'aspect ni les fonctionnalités souhaités.

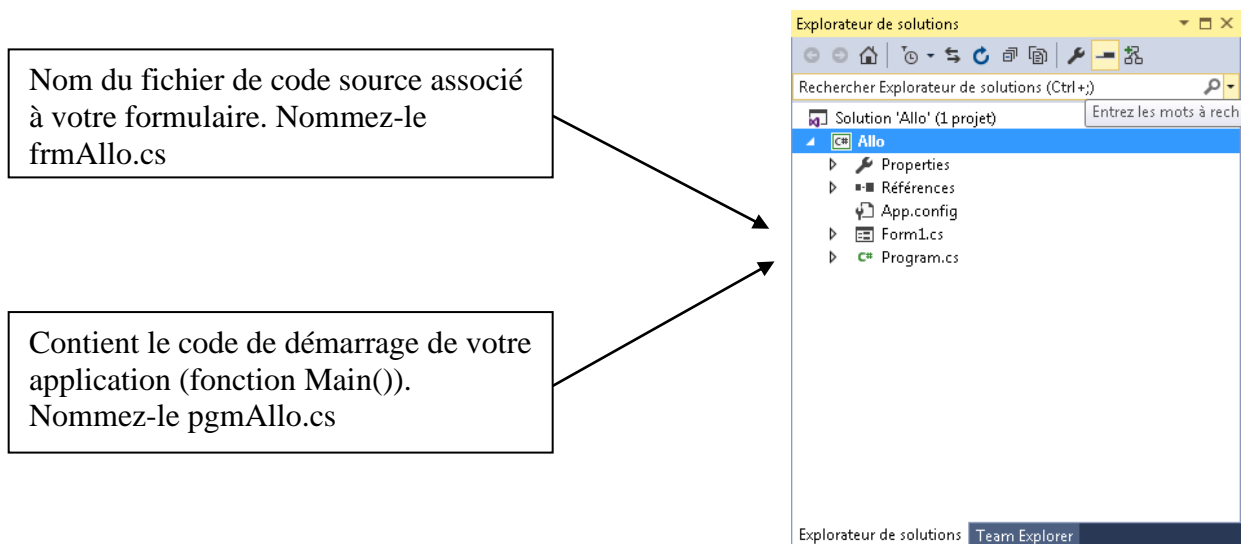
Vous allez utiliser les outils Visual Studio pour personnaliser le programme Allo.EXE jusqu'à ce qu'il apparaisse et fonctionne comme vous le désirez.

La conception visuelle (À faire)

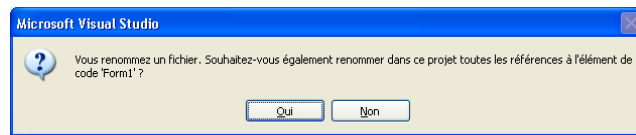
Comme nous l'avons vu, l'écriture d'un programme implique deux étapes: la conception visuelle et l'écriture du code.

Vous allez maintenant réaliser l'étape de conception visuelle. Vous allez utiliser les outils de Visual Studio pour personnaliser le formulaire principal du programme Allo.exe jusqu'à ce qu'il ait l'aspect souhaité.

- 1) Nommez les objets de façon significative. Dans l'explorateur de solution, donnez le nom du fichier source (par défaut, il s'appelle **Form1.cs**, modifiez-le pour **frmAllo.cs**)
Ce fichier contient les instructions de votre programme. Modifier aussi le nom : Program.cs par pgmAllo.cs (ce fichier contient le code de démarrage de l'application)



Lorsque vous recevrez la boîte de message suivante, Cliquer sur OUI pour que Visual Studio mette à jour toutes les références aux nouveaux noms.

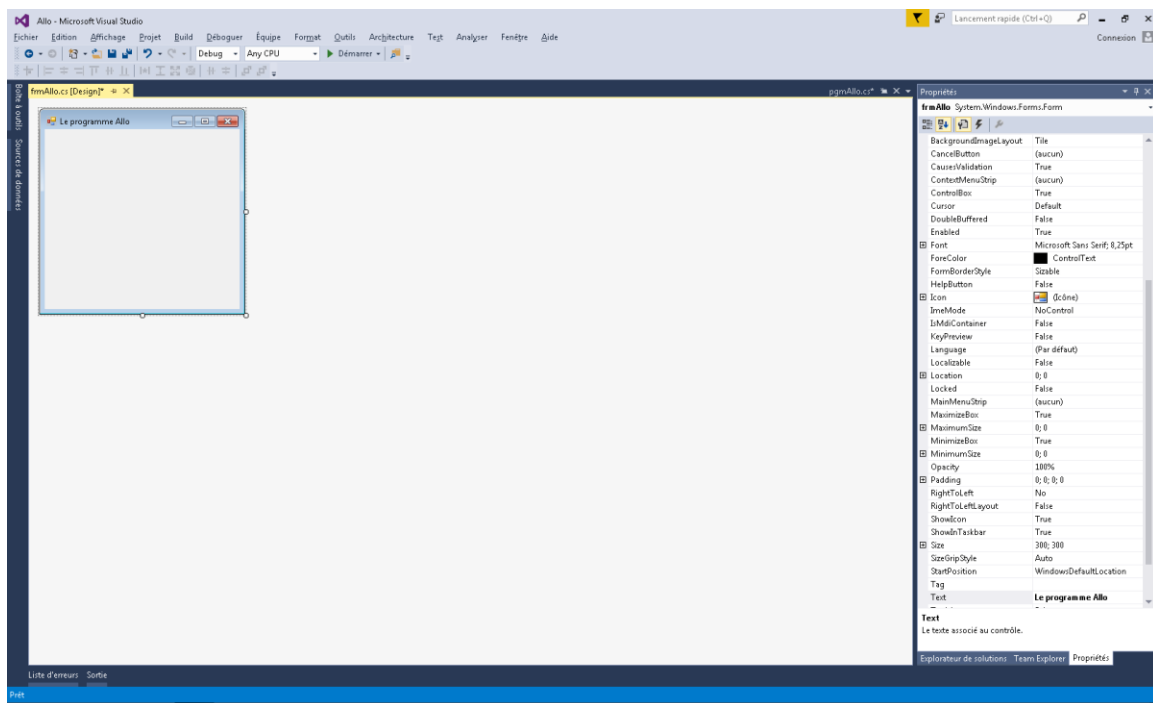


2) Sélectionner le formulaire dans l'onglet frmAllo.cs (design) et si les propriétés ne sont pas affichées, cliquer sur le bouton droit de la souris; faire afficher les propriétés du formulaire. Les propriétés s'affichent dans le coin inférieur droit de la fenêtre de Visual Studio.

Les propriétés d'un objet déterminent l'allure que prendra l'objet ainsi que ses fonctionnalités (couleur, dimension, apparence, visibilité, etc.)

Modifier la propriété Text du formulaire pour : Le programme Allo (c'est ce qui apparaîtra dans la barre de titre).

Vous pouvez aussi changer d'autres propriétés telles que BackColor et Icon



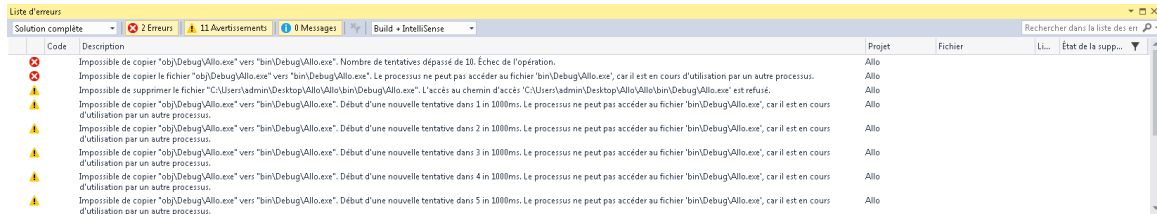
3) Générer votre projet.

4) Exécuter votre projet.

La seule chose qui devrait avoir changé est le contenu de la barre de titre et la couleur de fond si vous l'avez modifié.

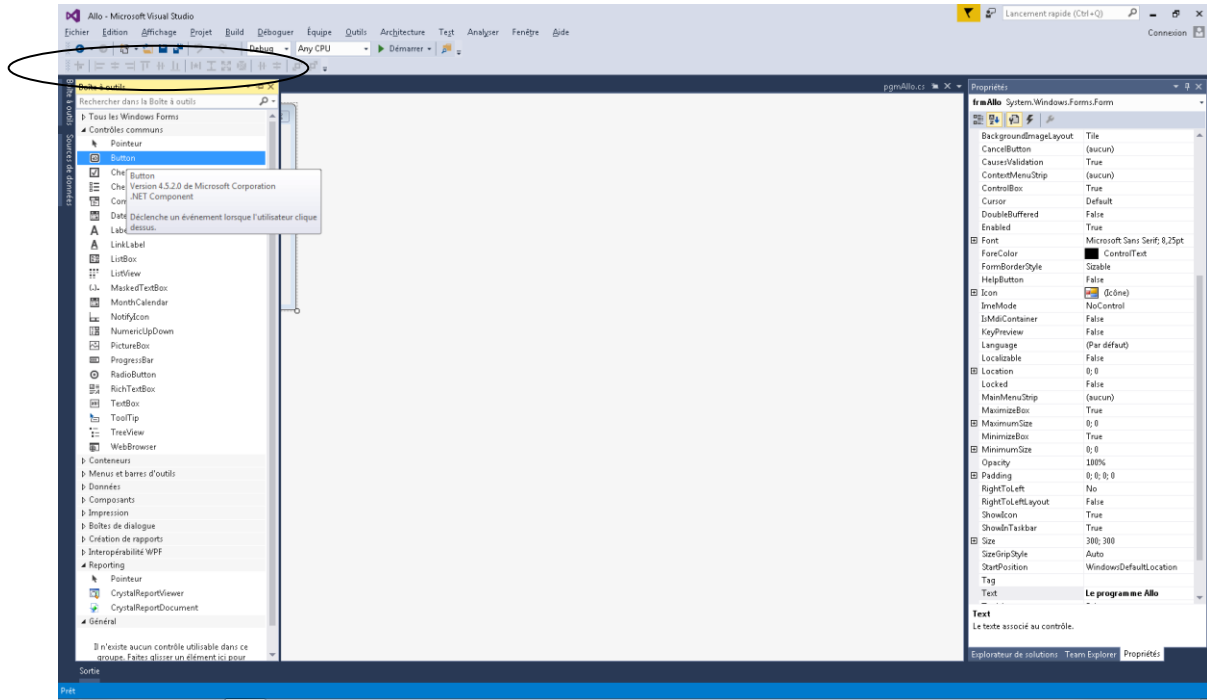
5) Fermer votre application.

⚠ L'application ne doit pas être en cours exécution lorsque vous compilez. Si c'est le cas, vous obtiendrez le message suivant :

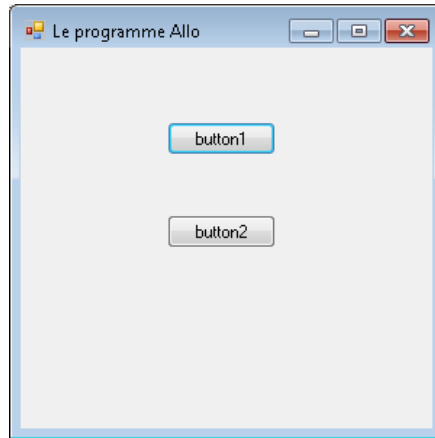


Tout ce que vous avez à faire à ce moment-là, c'est de fermer votre application (pas Visual Studio) qui doit se trouver minimisé dans la barre de tâche, et de recommencer la compilation.

Maintenant, nous allons placer les objets sur le formulaire. Nous voulons placer deux(2) boutons de commandes. Pour placer un objet (contrôle) sur le formulaire, il suffit d'ouvrir la boîte à outils identifiée par le pictogramme à gauche de l'environnement de développement.



Placer deux(2) boutons de commande (le contrôle Button) dans votre formulaire. Dimensionnez-les tels que désirés. Servez-vous de la barre d'outils (celle encadrée) au besoin pour dimensionner et ajuster les contrôles dans le formulaire.



Comme pour le formulaire, il faut changer les propriétés des objets placés sur le formulaire. Pour changer une propriété d'un objet, d'abord sélectionner l'objet (ne pas double-cliquer juste cliquer) puis dans l'onglet Propriétés, modifier les propriétés désirées.

Modifier la propriété Name et la propriété Text de chaque bouton.

Button1 :Propriété **Name** = btnAfficherAllo

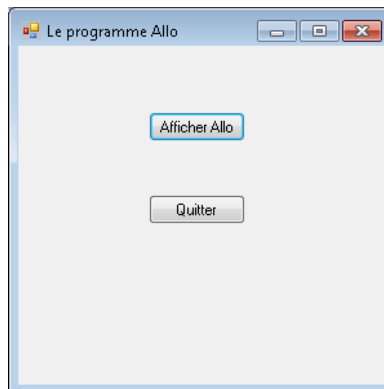
Propriété **Text** =Afficher Allo

Button2 :Propriété **Name** = btnQuitter

Propriété **Text** = Quitter

Vous pouvez changer à votre guise la propriété BackColor de chaque bouton.

À nouveau, générer la solution. S'il n'y a aucune erreur à la génération, faites-le exécuter. Le formulaire, tel que conçu, s'affichera.



Bravo ! Vous avez terminé l'implantation visuelle du programme Allo.

Fermez le programme Allo.EXE en cliquant sur l'icône de fermeture dans l'angle supérieur droit de la fenêtre programme.

Association de code au bouton de commande btnAfficherAllo

Comme vous l'avez vu, à ce stade rien ne se passe lorsque vous cliquez sur les boutons de commande Afficher Allo et Quitter. Vous allez maintenant attacher du code au bouton de commande btnAfficherAllo afin que le programme réalise ce que l'on attend de lui lorsque l'on clique sur ce bouton. La façon la plus rapide d'attacher des instructions de programmation (code) à l'événement Clic d'un bouton de commande est de sélectionner le bouton dans le formulaire en mode design et de double-cliquer sur ce bouton. Automatiquement le squelette de la méthode est créé dans le fichier de code source et l'assistant nous amène directement là où il faut placer les instructions.

```
private void btnAfficherAllo_Click(object sender, EventArgs e)
{
}
```

La méthode contient:

- 1) La ligne d'entête: **private void btnAfficherAllo_Click(object sender, System.EventArgs e)**
 - private signifie que cette méthode n'est accessible qu'à l'intérieur de la classe frmAllo;
 - void signifie que cette méthode ne retourne aucune valeur ;
 - btnAfficherAllo_Click est le nom de la méthode ;
 - Les paramètres de la méthode sont toujours entre parenthèses. Ici, la méthode reçoit deux (2) paramètres : l'appelant de la méthode et l'événement qui a provoqué l'appel.
- 2) L'accolade d'ouverture qui indique le début du code de la méthode;
- 3) L'accolade de fermeture vous indiquant la fin du code de la méthode;

Insérez l'instruction appropriée à l'intérieur des {} de votre méthode:

```
private void btnAfficherHello_Click(object sender, EventArgs e)
{
    MessageBox.Show( "Allo le monde!!!",
                    "Allo",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Exclamation);
}
```

Le code de la méthode `btnAfficherAllo_Click()` est très simple (une instruction):

MessageBox est une classe de l'espace de nom **System** qui contient la méthode statique **Show**. Cette méthode permet d'afficher un Message dans une petite fenêtre appelée Boîte à message.

- Le premier paramètre de la méthode Show est le message à afficher;
- Le deuxième paramètre est le titre de la boîte de message;
- Le troisième permet d'ajouter un bouton OK pour fermer la boîte de message;
- Le dernier spécifie l'icône à afficher dans la boîte de message.

Pour voir fonctionner le code que vous avez attaché au bouton de commande `btnAfficherAllo`, il faut générer la solution et l'exécuter! Comment fait-on déjà ?

Comme prévu, le programme `Allo.EXE` répond en affichant une boîte de message lorsque l'on clique sur le bouton Afficher Allo.

- Fermez la boîte de message ALLO en cliquant sur son bouton OK.
- Essayez de cliquer sur le bouton de commande Quitter.

Bien entendu, rien ne se passe lorsque vous cliquez sur le bouton de commande Quitter puisque vous n'y avez pas encore attaché de code. C'est ce que vous allez faire dans la section suivante. Terminez le programme `Allo.EXE` en cliquant sur l'icône x dans l'angle supérieur droit de la fenêtre programme.

Association de code au bouton de commande `btnQuitter`

Procédez vous-même à l'ajout du code sur le bouton Quitter : (comme vous avez fait pour le premier bouton)

```
private void btnQuitter_Click(object sender, System.EventArgs e)
{
    Application.Exit();
}
```

Exit() est une méthode statique de la classe Application. Elle termine l'application.

Essayez votre programme. En cliquant sur le bouton Quitter, le programme Allo.EXE doit se terminer, ce qui vous indiquera que le code que vous avez associé au bouton de commande Quitter fonctionne!

Documentation de votre programme

Il est important pour les personnes qui auront à lire le code de vos programmes et éventuellement auront à le modifier, de pouvoir facilement s'y retrouver. Chaque entreprise a ses propres règles de documentation de programme, il faut s'y conformer.

Dans le cadre du cours de Programmation I les règles du département seront très simples :

1^{ère} règle : Au début du fichier de code, indiquez en commentaire, votre nom, la date de conception du programme et en quelques mots décrivez l'application. Un commentaire sur plusieurs lignes est entouré des caractères /* en début et */ en fin de commentaire. Un commentaire sur une seule ligne ou même sur la fin d'une ligne est précédé de //.

2^{ème} règle : Avant chaque fonction que vous écrivez, indiquez un bref résumé de la fonction.


```
/* *****  
* Votre nom  
* La date  
* Premier programme en Visual C#  
***** */  
  
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Text;  
using System.Windows.Forms;  
namespace Allo  
{  
    public partial class frmAllo : Form  
    {  
        public frmAllo()  
        {  
            InitializeComponent();  
        }  
        // Fonction qui affiche un message  
        private void btnAfficherAllo_Click(object sender, EventArgs e)  
        {  
            MessageBox.Show("Allo le monde!!!",  
                            "Allo",  
                            MessageBoxButtons.OK,  
                            MessageBoxIcon.Exclamation);  
        }  
        // Fonction qui ferme le programme  
        private void btnQuitter_Click(object sender, EventArgs e)  
        {  
            Application.Exit();  
        }  
    }  
}
```

Générer votre programme et exécutez-le pour vous assurer que rien n'a changé à l'exécution.

Enregistrer un programme dans votre espace réseau.

Lorsque vous terminez un programme ou que vous arrêtez simplement de travailler, il est important de conserver **votre projet complet** sur un support qui vous est propre, d'où il pourra être récupéré facilement (votre espace réseau, une clé USB..). En effet, l'unité de disque D: sur votre ordinateur est accessible par tous les étudiants; les fichiers contenus sur D: peuvent être modifiés, détruits, récupérés par n'importe qui.

Pour copier votre projet:

- **fermer Visual Studio**
- **copier le dossier qui contient votre projet sur votre espace réseau ou votre clé USB.**
( tout le dossier Allo, celui du plus haut niveau)

D:\Votre Répertoire\Allo

- **Ensuite vous devez détruire votre dossier sur D:**

Récupérer un programme.

Pour récupérer un programme conservé sur votre espace réseau ou sur votre clé USB :

- Créer un répertoire personnel sur D: et copier tout le dossier de votre projet à partir de votre espace réseau ou votre clé USB .

Démarrer ensuite Visual Studio et ouvrez votre projet à partir de l'unité D:.

 **Ne travaillez jamais à partir de votre espace réseau ou de votre clé.**

Exercices d'approfondissement.

1. Créer un nouveau projet appelé Tigre:

Nom du projet: Tigre

Choisissez l'emplacement: sur D:\Votre dossier

2. **Nommez correctement les fichiers de code:**

- Changez le nom Program.cs par pgmTigre.cs
- Changez le nom Form1.cs par frmTigre.cs

3. Modifier les propriétés du formulaire pour:

- Afficher dans la barre de titre du formulaire: La gourmandise, c'est dangereux...
- Modifier la couleur du formulaire pour: PaleTurquoise

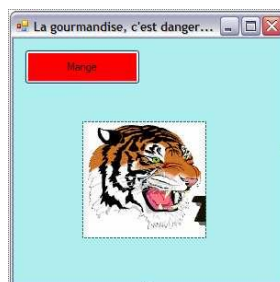
4. Dans le formulaire, ajouter un bouton de commande

- Nommez-le correctement (Propriété Name): btnMange
- Sur le bouton mettre le texte: Mange
- Modifier la couleur du bouton.

4. Dans le centre du formulaire, ajouter un objet PictureBox de la boîte d'outils à votre gauche

- Disposez l'objet au centre du formulaire et dimensionnez-le correctement
- Nommez l'objet (Propriété Name): picTigre
- Dans la propriété Image, attachez une image de tigre, vous devez télécharger une image de tigre sur le NET, un fichier avec extension .jpeg que vous aller déposer sur l'unité D: avant d'attache celle-ci à la propriété Image.
- modifier la propriété SizeMode pour lui attribuer la valeur: StretchImage dans les propriétés à votre droite.

Votre formulaire devrait avoir l'allure suivante:



6. Générez la solution et exécutez.

7. Attachez le code suivant au bouton btnMange:

```

private void btnMange_Click(object sender, EventArgs e)
{
    picTigre.Left = picTigre.Left -10;
    picTigre.Width = picTigre.Width + 20;
    picTigre.Height = picTigre.Height + 15;
    if (picTigre.Width >= Width)
    {
        MessageBox.Show("Attention, Explosion...");
        picTigre.Visible = false;
        BackColor = System.Drawing.Color.Red;
    }
}

```

8) Générez et exécutez votre programme.

9) Essayez de comprendre les instructions de la méthode btnMange_Click();

10) Documentez correctement votre programme. (Ajouter les commentaires)

11) Positionnez votre curseur à l'intérieur du mot MessageBox de votre fonction et appuyez sur la touche F1. Que se passe-t-il?

12) Fermer Visual Studio et copier votre projet sur votre espace réseau.

Exercice 2.

- 1) Récupérer votre projet Allo (Celui qui contient 2 boutons : le bouton Afficher Allo et le bouton Quitter).
- 2) Ajoutez-lui un bouton pour afficher 4 messages différents, de votre choix, en séquence. Utiliser des icônes différents dans la boîte de message.
- 3) Testez correctement.

Exercice 3.

Exécuter votre programme. Ne le fermez pas. Modifiez un de vos commentaires dans la fonction qui affiche 4 messages et générez la solution. Que se passe-t-il ?

Exercice 4.

Fermez votre application et Visual Studio.

Prenez le temps d'examiner les fichiers contenus dans le répertoire de votre projet.

Dans quel sous répertoire retrouvez-vous le programme exécutable de votre projet (le .exe)?

Quel est la taille de votre programme exécutable ? _____

Dans le dossier de votre projet sur D, il y a un sous-dossier (nom du projet) et un fichier de type Microsoft Visual Studio Solution (.sln) (qui porte aussi le nom de votre projet).

Que se passe-t-il lorsque vous double-cliquer sur ce fichier?

Montrez les exercices 2, 3 et 4 au prof.

Questions de révision

Sur quel unité de disque faut-il travailler et pourquoi ?

Lorsque vous sauvegardez votre projet, il est important de sélectionner le bon répertoire à sauvegarder : sur quel répertoire ?

Pour modifier une propriété d'un contrôle : il faut le sélectionner en cliquant ou en double cliquant ?

Pour associer du code à un bouton : il faut le sélectionner en cliquant ou en double cliquant ?

Si l'on désire obtenir de l'aide sur une méthode, que faut-il faire ?

Que se passe-t-il lorsque vous double cliquez sur un fichier dont l'extension est .sln ?

Y a-t-il une différence si vous double cliquez sur un fichier dont l'extension est .cs ? Laquelle ?

12 Travaux pratiques

12.1 Travail Pratique No. 1 : Les propriétés des contrôles

Objectifs spécifiques :

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de création d'un programme C#;
- manipuler les propriétés des contrôles;
- se conformer aux règles d'attribution de noms aux contrôles;
- associer une méthode suite à un événement sur un contrôle;
- expérimenter les étapes d'écriture d'un programme: codage, compilation, exécution et vérification;
- corriger les erreurs de compilation;
- sauvegarder un projet;
- récupérer un projet.

Point:

Correction :

Directives:

1. Créer un projet Visual C#

Modèle: Application Windows

Emplacement obligatoire : sur le D : dans un répertoire portant votre nom

Nom: TP1

Dans l'explorateur de solution, modifier le nom du fichier de code source Form1.cs pour frmTP1.cs

2. Propriétés du formulaire:

- a) Name: frmTP1
- b) Text: TP 1 - Les propriétés
- c) BackColor – Choisir Vert (128;255;128)
- d) Font: Courier New; 8,25pt

Compiler votre projet : Menu Générer; Générer la solution

Sauvegarder votre projet. Menu Fichier; Enregistrer tout

Exécuter votre projet: Menu Débuguer; Exécuter sans débogage

3. Ajouter 6 boutons à votre formulaire à l'aide de la boîte à outils. On fait un clic droit sur le contrôle pour accéder aux propriétés.

Propriété du premier bouton:

Name: btnAgrandirFormulaireEnLargeur

Text: Agrandir le formulaire en largeur

BackColor: Choisir la deuxième teinte de rouge (255; 128; 128)

Font: Lucida Console; 8,25pt

Propriété du deuxième bouton:

Name: btnAgrandirFormulaireEnHauteur

Text: Agrandir le formulaire en hauteur

BackColor: à votre choix

Font: à votre choix

Propriété du troisième bouton:

Name: btnDeplacerFormulaireVersDroite

Text: Déplacer le formulaire vers la droite

BackColor: à votre choix

Font: à votre choix

Propriété du quatrième bouton:

Name: btnDeplacerFormulaireVersGauche

Text: Déplacer le formulaire vers la gauche

BackColor: à votre choix

Font: à votre choix

Propriété du cinquième bouton:

Name: btnDeplacerFormulaireVersBas

Text: Déplacer le formulaire vers le bas

BackColor: à votre choix

Font: à votre choix

Propriété du sixième bouton:

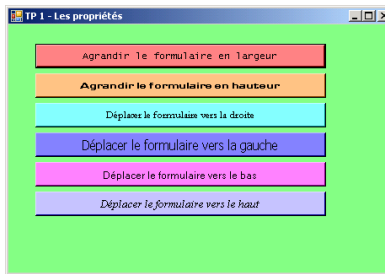
Name: btnDeplacerFormulaireVersHaut

Text: Déplacer le formulaire vers le Haut

BackColor: à votre choix

Font: à votre choix

4. En utilisant les outils de Visual Studio, aligner vos boutons et ajuster leur dimension.
Votre formulaire devrait ressembler à celui-ci.



5. Enregistrer votre projet (enregistrer tout) et faire vérifier par le professeur.
6. Ajouter la méthode Click sur le premier bouton en double-cliquant sur le bouton. Si la propriété Name du bouton a été changée avant le double-clic, le nom de la méthode inclura le nom de la propriété Name. Il est très complexe de changer le nom du bouton après avoir ajouté sa méthode Click.

```
private void btnAgrandirFormulaireEnLargeur_Click(object sender, EventArgs e)
{
    Width = Width + 5;
}
```

La propriété Width représente la largeur de l'objet courant c'est-à-dire le formulaire. La méthode augmente donc la largeur du formulaire de 5 pixels (plus petit point lumineux).

Compiler, exécuter et constater le résultat.

7. En examinant les propriétés du formulaire, trouvez la propriété qui représente la hauteur du formulaire.
Comment accéder aux propriétés du formulaire ? _____
Quel est le nom de la propriété qui représente la hauteur du formulaire? _____

Ajouter la méthode Click sur le deuxième bouton pour augmenter la hauteur du formulaire de 10 pixels.

Compiler, exécuter et constater le résultat.

8. Ajouter la méthode suivante sur le troisième bouton.

```
private void btnDeplacerFormulaireVersDroite_Click(object sender, EventArgs e)
{
    Left = Left + 10;
}
```

La propriété Left représente la coordonnée X du bord gauche (x,y) du formulaire. La méthode augmente cette propriété de 10 pixels donc un déplacement du formulaire vers la droite. Pourquoi vers la droite ? _____

13. Ajouter un huitième bouton sur le formulaire : Name: btnBouton8
Text: Réduire Bouton 6

Ajouter la méthode Click du bouton 8 pour réduire la dimension du bouton 6 (largeur et hauteur).

Quelle instruction faut-il inscrire ?

Compiler, exécuter et constater le résultat.

- [illegible]

Ajouter la méthode suivante sur le click du bouton 9.

```
private void btnBouton9_Click(object sender, EventArgs e)
{
    BackColor = System.Drawing.Color.Peru;
}
```

La propriété `BackColor` représente la couleur de fond du contrôle. La couleur `Peru` est l'une des couleurs disponibles dans la structure `Color` de l'espace de nom `System.Drawing`. En tapant le point qui suit `Color`, vous aurez une liste qui vous aide à faire votre choix.

Compiler, exécuter et constater le résultat.

- [illegible]

Ajouter la méthode sur le click du bouton 10 pour modifier la couleur du bouton 6 en "Purple".

Quelle instruction faut-il inscrire ?

Compiler, exécuter et constater le résultat.

Enregistrer votre projet (enregistrer tout) et faire vérifier votre projet et votre cahier par le professeur.

16. Ajouter un onzième bouton sur le formulaire : Name: btnBouton11
Texte: Choisir la couleur du formulaire

Ajouter la méthode suivante sur le click du bouton 11.

```
private void btnBouton11_Click(object sender, EventArgs e)
{
    ColorDialog MaCouleur = new ColorDialog();
    if (MaCouleur.ShowDialog() == DialogResult.OK)
    {
        BackColor = MaCouleur.Color;
    }
}
```

Dans cette méthode, nous déclarons une nouvelle instance (MaCouleur) de la classe ColorDialog. On appelle ensuite la méthode ShowDialog() sur cette instance ce qui fait afficher le dialogue pour choisir la couleur. Si l'utilisateur quitte le dialogue en appuyant sur le bouton OK, la couleur sélectionnée (MaCouleur.Color) est appliquée à la propriété BackColor du formulaire.

Compiler, exécuter et constater le résultat.

Notez que la couleur s'applique à tous les objets du formulaire auxquels nous n'avons pas appliqué spécifiquement une couleur. Les objets héritent automatiquement de la couleur du Parent.

17. Ajouter un douzième bouton sur le formulaire : Name: btnBouton12
Text: Choisir la couleur du bouton 6

Ajouter la méthode Click du bouton 12 pour faire choisir une couleur et l'appliquer sur le bouton 6.

Quelle instruction faut-il inscrire ? _____
Compiler, exécuter et constater le résultat.

18. Ajouter un treizième bouton sur le formulaire : Name: btnBouton13
Text: Choisir la police du formulaire

Ajouter la méthode suivante sur le click du bouton 13.

```
private void btnBouton13_Click(object sender, EventArgs e)
{
    FontDialog MaPolice = new FontDialog();
    if (MaPolice.ShowDialog() == DialogResult.OK)
    {
        Font = MaPolice.Font;
    }
}
```

Dans cette méthode, nous déclarons une nouvelle instance (MaPolice) de la classe FontDialog. On appelle ensuite la méthode ShowDialog() sur cette instance ce qui fait afficher le dialogue pour choisir la police. Si l'utilisateur quitte le dialogue en appuyant sur le bouton OK, la police sélectionnée (MaPolice.Font) est appliquée à la propriété Font du formulaire.

Compiler, exécuter et constater le résultat.

Notez que la police s'applique à tous les objets du formulaire auxquels nous n'avons pas appliqué spécifiquement une police. Les objets héritent automatiquement de la police du Parent.

19. Ajouter un 14^{ième} bouton sur le formulaire : Name: btnBouton14
Text: Choisir la police du bouton 6

Ajouter la méthode sur le click du bouton 14 pour faire choisir une police et l'appliquer sur le bouton 6.

Compiler, exécuter et constater le résultat.

20. Ajouter un 15^{ième} bouton sur le formulaire :Name: btnBouton15
Text: Faire disparaître le bouton 7

Ajouter la méthode suivante sur le click du bouton 15.

```
private void btnBouton15_Click(object sender, EventArgs e)
{
    if (btnBouton7.Visible)
    {
        btnBouton15.Text = "Faire apparaître le bouton 7";
        btnBouton7.Visible = false;
    }
    else
    {
        btnBouton15.Text = "Faire disparaître le bouton 7";
        btnBouton7.Visible = true;
    }
}
```

Le code de cette méthode utilise une structure alternative. Si la propriété Visible du bouton 7 est actuellement à l'état Vrai (true), nous rendons le bouton invisible en mettant cette propriété à Faux (false) et on change la propriété Text du bouton 15 pour que sa nouvelle signification soit exacte.

Dans le cas contraire, c'est-à-dire que le bouton 7 est actuellement invisible, nous le rendons visible en mettant la valeur Vrai (true) à cette propriété et nous changeons le texte du bouton 15.

Compiler, exécuter et constater le résultat.

21. Ajouter un 16^{ième} bouton sur le formulaire :Name: btnBouton16
Text: Désactiver le bouton 7

Ajouter le code sur le bouton 16 pour faire le même traitement qu'en 20 mais avec la propriété Enabled.

Que se passe-t-il lorsque l'on change l'état de la propriété Enabled ?

Compiler, exécuter et constater le résultat.

22. Inclure en commentaire dans votre code source avant la première méthode que vous avez écrite, votre nom, la date de création et un commentaire général de l'application. Enregistrer votre projet. Quitter Visual Studio.

Copier votre projet complet dans votre espace disque sur le serveur. Faire vérifier votre projet et votre cahier par le prof.

12.2 Travail Pratique No. 2 : La structure séquentielle

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes;
- concevoir une application en Visual C#;
- pratiquer l'utilisation des structures séquentielles et des notions de traduction en langage C #;
- utiliser les types de données en C#;
- concevoir les données de test afin de valider les résultats de l'application.

Date de remise:

Point:

Correction :

Directives:

Débuter un nouveau projet Windows en C# en respectant les consignes suivantes :

Nom du projet TP2

Nom du formulaire frmTP2

Titre du formulaire TP2 par (votre nom)

Déposer 2 boutons « Caluler l'escompte » et « Calculer Hydro-Cégep » sur votre formulaire et les nommer **btnCalculerEscompte** et **btnCalculerHydro** via la propriété Name.

Le bouton **btnCalculerEscompte** :

Traduire en C# l'algorithme de l'exercice 11 de 2.5

Ajouter les boîtes d'édition nécessaire pour vos entrées et vos sorties en les nommant de façon significative. Programmer le bouton **btnCalculerEscompte** et le tester avec le jeu d'essais.

Le bouton **btnCalculerHydro** :

Appliquer la démarche en huit (8) étapes du point 2.1 pour programmer le bouton btnCalculerHydro à partir d'informations suivantes : (N.B. L'écrit de votre démarche sera vérifié par le professeur.)

La Société Hydro-Cégep vous demande de concevoir une application qui permet de calculer le détail de la facture d'un client selon sa consommation.

L'application devra répondre aux consignes suivantes.

- **Dans la barre de titre, on retrouve: Facturation HYDRO-CÉGEP;**
- **Les zones d'édition doivent être clairement identifiées;**
- **Le bouton btnCalculerHydro déclenche la facturation du client.**

Énoncé du problème:

À partir du numéro de l'abonné (composé de lettres et de chiffres), du nombre de jours facturés et du nombre de kilowatts consommés, calculer et afficher le montant de la facture en sachant que chaque abonné doit payer une redevance d'abonnement de 0.25 \$ par jour; de plus, on facture 10 cents par kilowatts pour les 100 premiers kilowatts et 20 cents par kilowatt pour les kilowatts excédentaires. Calculer et afficher aussi une taxe de 10 % de même que le total à payer.

Notez bien que tous les clients ont toujours une consommation supérieure à 100 kilowatts.

12.3 Travail Pratique No. 3 : Les structures séquentielles et alternatives

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes
- concevoir une application en Visual C#
- pratiquer l'utilisation des structures séquentielles et des structures alternatives.
- utiliser les notions de traduction en langage C#.
- utiliser les types de données
- concevoir les données de test afin de valider les résultats de l'application.

Date de remise:

Point:

Correction :

Directives: Faire l'analyse, la conception visuelle, l'algorithme, les jeux d'essais et la trace avant le début du prochain laboratoire. L'élaboration de l'application sur ordinateur se fera au prochain laboratoire. Votre travail sur papier sera vérifié.

Un professeur vous demande de concevoir une application qui lui permettra de calculer la note finale des étudiantes en Bureautique.

L'application devra répondre aux consignes suivantes.

- Être développée sous la forme d'un formulaire Windows
- Dans la barre de titre, on retrouvera: Calcul des notes finales COURS BUREAUTIQUE
- Les zones d'édition doivent être clairement identifiées.
- Le formulaire doit être muni d'un bouton pour déclencher le calcul.

Énoncé du problème:

Le professeur saisi le nom de l'étudiante ainsi que son numéro matricule, la note de chacun des trois examens et la notes de chacun des quatre travaux. Les examens 1 et 2 sont comptabilisés sur 20 points; l'examen 3 sur 30 points. Les travaux 1 et 3 sont sur 20 points, le travail 2 sur 10 points et le travail 4 sur 15 points. Il y a possibilité d'une partie fractionnaire pour chaque travaux et examen. Le calcul de la note finale sur cent comprend les examens pour 80 % et les travaux pour 20%.

De plus votre application doit afficher
la mention "RÉUSSITE" pour une note finale supérieure ou égale à 60 %
ou la mention "ÉCHEC" pour une note inférieure à 60 %.

La note finale doit être arrondie à 2 décimales.

Essai :

Nom: Sylvie Côté

Matricule : AA07

Examen 1 : 18

Examen 2 : 15.5

Examen 3 : 22.5

Travail 1 : 15

Travail 2 : 6

Travail 3 : 2

Travail 4 :

15

Résultat : 75.69 Réussite

Si votre résultat n'arrive pas exactement à 75.69, revoyez votre formule de calcul. Chaque examen a sa propre pondération. L'ensemble des notes d'examens compte pour 80 % de la note finale. L'ensemble des notes pour les travaux compte pour 20%.

12.4 Travail Pratique No. 4 : Les structures séquentielles et alternatives

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes
- concevoir une application en Visual C#
- pratiquer l'utilisation des structures séquentielles et des structures alternatives.
- Utiliser les variables compteurs et les accumulateurs.
- Utiliser des variables membres
- Découper le traitement
- Valider les données d'entrée.
- utiliser les notions de traduction en langage C#
- utiliser les types de données
- concevoir les données de test afin de valider les résultats de l'application.

Date de remise:

Point:

Correction :

Directives: Faire l'analyse, la conception visuelle, l'algorithme, les jeux d'essais et la trace avant le début du prochain laboratoire. L'élaboration de l'application sur ordinateur se fera au prochain laboratoire. Votre travail sur papier sera vérifié.

Énoncé :

Luigi, le propriétaire d'une célèbre pizzeria, veut informatiser son entreprise. Dans un premier temps, il désire calculer automatiquement le prix de ses produits. Il vend une pizza à la fois. Lors d'une vente, il entre la grandeur de la pizza ainsi que le nombre d'extras. Il veut obtenir le prix de vente de cette pizza. Tout au long de la journée, il veut savoir le nombre de pizzas (par grandeur) qu'il a vendu et le total d'argent en caisse. Chez Luigi, on vend 3 grandeurs de pizzas: la "petite" (10 pouces de diamètre), la "médium" (12 pouces) et la "large" (16 pouces). On peut l'obtenir "ordinaire" (sauce et fromage seulement) ou avec des extras (peppéroni, champignons, oignons).

Le prix de vente de la pizza dépend de sa grandeur et du nombre d'extras. **Le prix de vente est 1.5 fois le coût de production.**

Le coût de production comprend un coût de base fixe, un coût de base variable qui dépend de la grandeur de la pizza (plus précisément coût de base variable X la surface de la pizza) et un coût variable additionnel pour chaque extra. Luigi nous a dit que tous ses extras lui coûtent le même prix...!

On peut résumer les **coûts de production** avec la formule suivante:

Coût de production = Coût fixe + (coût de base X la surface) + (Le nombre d'extras X le cout extra X la surface)

- les coûts fixes sont de 3.00
- le coût de base est de 0.035
- le coût d'extra est de 0.006

- La surface = π (grandeur / 2)²

$$\pi = 3.1416$$

Vous devez **valider** les données d'entrée: Pour la grandeur, on accepte 10, 12 ou 16 seulement.
Pour les extras, on accepte les valeurs entre 0 et 3 inclusivement.

Pour vous aider à vérifier vos résultats, voici la liste des **prix de ventes** pour 1 pizza.

Grandeur \ Extra	Extra			
	0	1	2	3
petite	8.62	9.33	10.04	10.74
médium	10.44	11.46	12.47	13.49
large	15.06	16.87	18.67	20.48

12.5 Travail Pratique No. 5 : Les structures séquentielles et alternatives

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes
- pratiquer l'utilisation des structures séquentielles et des structures alternatives complexes et imbriquées.
- utiliser les notions de traduction en langage C#
- utiliser les types de données
- différencier l'utilisation des variables locales et membres
- distinguer les niveaux de traitement.

Date de remise:

Point:

Correction :

Directives: Faire l'analyse, la conception visuelle, l'algorithme, les jeux d'essais et la trace avant le début du prochain laboratoire. L'élaboration de l'application sur ordinateur se fera au prochain laboratoire. Votre travail sur papier sera vérifié.

Énoncé :

La compagnie de transport **Aérodrome Pout Pout** vous demande de concevoir une application, en Visual C#, qui lui permettra d'émettre ses billets de transport. **Vous devez respecter la conception visuelle déjà élaborée.**

Pour émettre un billet, le (la) commis qui se servira de votre application n'aura qu'à saisir le nom du voyageur; le code de destination soit M pour Montréal, T pour Toronto, V pour Vancouver; le code de sexe du voyageur soit F pour Féminin ou M pour Masculin; le code de statut civil du voyageur soit M pour Marié ou C pour Célibataire; et, finalement, le poids des bagages du voyageur. Ensuite, en appuyant sur le bouton Émission du billet, votre application devra produire le billet dans lequel on retrouvera:

Une formule de politesse:	M. pour un homme
	Mme pour une femme mariée
	Mlle pour une femme célibataire

Le nom du voyageur
La ville de départ qui est toujours Québec
La ville de destination
Le prix du billet

Le prix du billet comprend:

Le coût du siège soit 100 \$ pour Montréal
 200 \$ pour Toronto
 400 \$ pour Vancouver

Plus le coût des bagages, à savoir :0,50\$ du Kilo pour les 25 premiers Kilos et 0,75\$
du Kilo pour les Kilos excédentaires.

Et enfin, la compagnie donne une réduction de 10 % pour les billets excédant 500 \$.

Afin d'éviter les erreurs de saisie, vous devez valider les codes de destination, de sexe et de statut.
Validez aussi le poids des bagages afin qu'il ne puisse pas être négatif.

De plus, à la fin du traitement, votre programme doit calculer et afficher le pourcentage des billets émis pour chacune de destinations et la moyenne des prix de vente des billets qui ont été émis pour les femmes mariées. Ces valeurs doivent être arrondies à 1 décimale.

Jeu d'essais:

Nom du voyageur	Destination	Sexe	Statut	Poids des bagages
Guy Lajoie	M	M	M	15,5
Lily Brown	T	F	M	25
Joe Blo	V	M	C	35
Lise Roy	V	F	C	200
Marie Pinel	T	F	M	30
Lucie Labrie	V	F	M	100

Conception Visuelle:

Première saisie:

Transport Air-Cégep

Saisir les données du voyageur

Nom:

Sexe (M ou F) Statut (M ou C)

Destination (M, T ou V)

Poids des bagages:

M.

De Québec à:

Prix du Billet: \$

Émission du billet et les saisies suivantes:

Transport Air-Cégep

Saisir les données du voyageur

Nom:

Sexe (M ou F) Statut (M ou C)

Destination (M, T ou V)

Poids des bagages:

Mme

De Québec à:

Prix du Billet: \$

Lorsqu'on appuie sur le bouton statistique:

Transport Air-Cégep

Statistiques

Pourcentage des billets émis pour:

Montréal:	<input type="text" value="50"/>	%
Toronto:	<input type="text" value="50"/>	%
Vancouver:	<input type="text" value="0"/>	%

Moyenne des prix de ventes des femmes mariées:

\$

12.6 Travail Pratique No. 6 : Les structures séquentielles, alternatives et répétitives

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- Consolider l'utilisation de la structure répétitive.

Date de remise:

Point:

Correction :

Directives: Faire les étapes complètes pour résoudre chacun des problèmes suivants dans le formulaire : un bouton pour chaque problème.

1. Bouton « Somme 1..100 »

Calculer la somme des entiers de 1 à 100. Afficher le résultat dans une boîte à message.

2. Bouton « Somme X..Y »

Calculer la somme des entiers compris entre deux nombres entiers positifs non nuls saisis par l'utilisateur. Valider les nombres lus. Afficher le résultat dans une boîte à message.

3. Bouton « Puissance »

Lire deux nombres entiers positifs non nuls, valider les nombres lus et calculer le premier à la puissance du second sans utiliser la méthode Math.Pow. Afficher le résultat dans une boîte à message.

4. Bouton « Décompte »

Lire deux (2) nombres entiers négatifs ou nuls, valider les nombres et afficher dans un contrôle de type ListBox tous les nombres compris entre le plus grand et le plus petit.

5. Bouton « Étoiles »

Ajouter des étoiles (*) dans un contrôle de type ListBox à chaque fois que vous appuyez sur le bouton. Chacune des lignes supplémentaires compte une étoile additionnelle par rapport à la ligne précédente.

6. Bouton « Répète »

Lire un nombre entier et un nombre de répétitions supérieur à 1. Afficher ce nombre dans un contrôle de type TextBox autant de fois que lu en entrée. Par exemple : pour un nombre lu à 3 et un nombre de répétition lu à 5, la chaîne 3 3 3 3 3 est affichée.

7. Bouton « Renversant »

Lire un nombre entier positif supérieur ou égal à 10 et l'afficher à l'envers. Par exemple : le nombre lu à 12345, le résultat affiché sera 54321. Vous devez utiliser la division et le modulo donc travailler sur un nombre entier non pas sur une chaîne.

Faire vérifier par le prof.

12.7 Travail Pratique No. 7 : Les structures séquentielles, alternatives et répétitives

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- Consolider l'utilisation de la structure répétitive.

Directives:



Le chiffrement de César

Ce que l'on appelle le chiffrement de César est probablement l'un des plus anciens codages au monde (et plus certainement l'un des plus simples qui soient), dans la mesure où Jules César lui-même l'aurait utilisé.

L'antiquité – les Romains (Il y a 2050 ans)

Cryptographie : le chiffre de César

– Chiffrement par substitution simple

- Le secret est dans une lettre qui indique un décalage (de 3 positions en principe)

ABCDEF GHI J KLMNOPQRSTUVWXYZ
DEFGH I JKLMNOPQRSTUVWXYZABC

Message en clair

bob m'entends tu ?

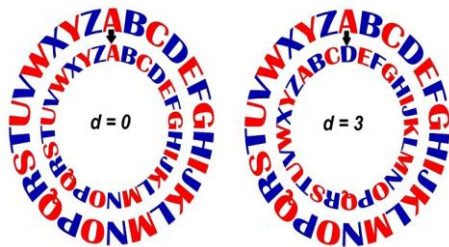
Message chiffré

ERE P'HQWHOGV WX ?



Aussi appelé chiffrement par décalage, il consiste simplement en une permutation de chaque lettre par une autre, par translation d'un certain nombre de positions dans l'alphabet (toujours dans le même sens bien sûr). Si l'on fait un décalage à droite de trois positions du mot CESAR, cela donne FHVDU (car $C+3=F$ dans l'alphabet).

Ce chiffrement par substitution est donc une simple permutation circulaire de l'alphabet qui peut s'exprimer à l'aide d'une congruence sur les entiers. Prenons l'entier n comme clé de cryptage.



La clé de cryptage est le caractère correspondant à la valeur que l'on ajoute à chaque lettre pour effectuer le codage. Dans le premier exemple, la clé est C (étant la 3ème lettre de l'alphabet).

Ce système de cryptage symétrique a pour inconvénient d'être particulièrement simple à casser, une soustraction permettant de remonter à la lettre substituée. Afin de connaître la clé de cryptage, il suffit d'une petite étude statistique. En effet, certaines lettres sont plus fréquentes que d'autres : en français par exemple, c'est la lettre « e » qui revient le plus souvent. Ainsi, la lettre étant la plus fréquente dans le message à décoder, peut correspondre au « e ». Il ne reste plus ensuite qu'à décrypter le reste du message.

Exemple : <https://www.dcode.fr/chiffre-cesar>

Message à envoyer : VIENSMEREJOINDREALASALLEDEBAIN

Chiffre de César 3 soit un **Décalage de 3** vers la droite

Alphabet
ABCDEFGHIJKLMNOPQRSTUVWXYZ

Alphabet décaler
DEFGHIJKLMNOPQRSTUVWXYZABC

Donne : **Message encrypté**

YLHQVPHUHMRLQGUHDODVDOOHGHEDLQ

The screenshot shows a web application titled 'Code de César'. At the top, there is a text input field labeled 'Code Cesar' with the value '3'. Below this, the application displays two alphabets: 'Alphabet' (ABCDEFGHIJKLMNOPQRSTUVWXYZ) and 'Alphabet Décalé' (DEFGHIJKLMNOPQRSTUVWXYZABC). There are three main sections for text processing: 1. Encoding: 'Phrase à Coder' (VIENSMEREJOINDREALASALLEDEBAIN) and 'Phrase Coder' (YLHQVPHUHMRLQGUHDODVDOOHGHEDLQ) with a 'Phrase a coder' button. 2. Decoding: 'Phrase Coder' (YLHQVPHUHMRLQGUHDODVDOOHGHEDLQ) and 'Phrase Décoder' (VIENSMEREJOINDREALASALLEDEBAIN) with a 'Phrase à décoder' button. The interface is clean and uses a light blue and grey color scheme.

Travail Pratique No. 8 : Les Tableaux

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- Manipuler les tableaux à une dimension;
- Consolider l'utilisation de la structure répétitive.

Date de remise:

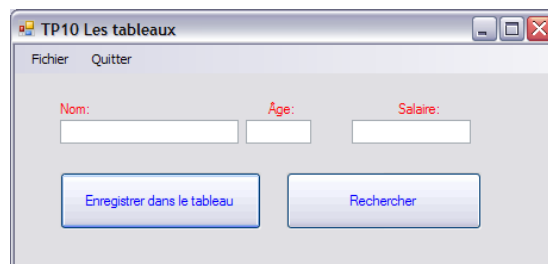
Point:

Correction :

Directives:

Copier le dossier Tp7 Tableaux à partir du serveur dans votre répertoire sur D:

1. Exécuter et Comprendre le fonctionnement.



// Déclaration des variables membres

```
string[] m_TNom = new string[100];
int[] m_TAge = new int[100];
double[] m_TSalaire = new double[100];
int m_Compteur;
```

// Déclaration des menus et boutons

```
private System.Windows.Forms.MenuStrip menuStrip1;
private System.Windows.Forms.ToolStripMenuItem mnuFichier;
private System.Windows.Forms.ToolStripMenuItem mnuChargerTableau;
private System.Windows.Forms.ToolStripMenuItem mnuEnregistrerTableau;
private System.Windows.Forms.ToolStripMenuItem mnuQuitter;
private System.Windows.Forms.Button btnEnregistrer;
private System.Windows.Forms.Button btnRechercher;
```

// La méthode constructeur

```
public frmTP10Tableaux()
{
    InitializeComponent();
    m_Compteur = 0;
}
```

// Méthode qui enregistre les données dans les tableaux

```
private void btnEnregistrer_Click(object sender, System.EventArgs e)
```

```

{
    int Age;
    double Salaire;
    string cs;
    bool Resultat;
    if (m_Compteur == 100)
    {
        MessageBox.Show("Tableau rempli", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        return;
    }
    if (txtNom.Text == "")
    {
        MessageBox.Show("Le nom est obligatoire", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtNom.Focus();
        return;
    }
    Resultat = Int32.TryParse(txtAge.Text, out Age);
    if(Resultat == false)
    {
        MessageBox.Show("Entrez un nombre entier", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtAge.Focus();
        return;
    }
    if (Age < 15 || Age > 90)
    {
        MessageBox.Show("Âge invalide (entre 15 et 90)", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtAge.Focus();
        return;
    }
    Resultat = Double.TryParse(txtSal.Text, out Salaire);
    if(Resultat == false)
    {
        MessageBox.Show("Entrez un nombre réel", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtSal.Focus();
        return;
    }
    if (Salaire < 0)
    {
        MessageBox.Show("Salaire invalide (doit être positif)", "Bouton Enregistrer" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtSal.Focus();
        return;
    }
    m_TNom[m_Compteur] = txtNom.Text;
    m_TAge[m_Compteur] = Age;
    m_TSalaire[m_Compteur] = Salaire;
    m_Compteur++;
    cs = string.Format("Employé #{0} Inscrit: Nom: {1:s} Âge: {2:d} Salaire: {3:C}",
        m_Compteur, m_TNom[m_Compteur-1], m_TAge[m_Compteur-1],
        m_TSalaire[m_Compteur-1]);
    MessageBox.Show(cs);
}

```

// Méthode qui recherche un élément à partir d'un nom

```

private void btnRechercher_Click(object sender, System.EventArgs e)
{
    int Ind = 0;

    if (txtNom.Text == "")
    {
        MessageBox.Show("Le nom est obligatoire", "Bouton Rechercher" ,
            MessageBoxButtons.OK, MessageBoxIcon.Stop);
        txtNom.Focus();
        return;
    }
}

```

```
    }  
    while (Ind < m_Compteur && txtNom.Text == m_TNom[Ind])  
        Ind++;  
    if (txtNom.Text == m_TNom[Ind])  
    {  
        txtAge.Text = m_TAge[Ind].ToString();  
        txtSal.Text = m_TSalaire[Ind].ToString();  
    }  
    else  
        MessageBox.Show("Employé Inexistant");  
}
```

// Méthode pour enregistrer les données des tableaux dans un fichier

```
private void mnuEnregistrerTableau_Click(object sender, System.EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();

    saveFileDialog1.InitialDirectory = "f:\\\" ;
    saveFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*" ;
    saveFileDialog1.FilterIndex = 2 ;
    saveFileDialog1.RestoreDirectory = true ;

    if(saveFileDialog1.ShowDialog() != DialogResult.OK)
        return;
    FileStream fs = new FileStream(saveFileDialog1.FileName, FileMode.Create);
    BinaryWriter w = new BinaryWriter(fs);
    w.Write( "Fichier Employé");
    w.Write(m_Compteur);
    for (int i = 0; i < m_Compteur; i++)
    {
        w.Write(m_TNom[i]);
        w.Write(m_TAge[i]);
        w.Write(m_TSalaire[i]);
    }
    w.Close();
    fs.Close();
}
```

// Méthode pour remplir les tableaux à partir d'un fichier

```
private void mnuChargerTableau_Click(object sender, System.EventArgs e)
{
    OpenFileDialog openFileDialog1 = new OpenFileDialog();

    openFileDialog1.InitialDirectory = "f:\\\" ;
    openFileDialog1.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*" ;
    openFileDialog1.FilterIndex = 2 ;
    openFileDialog1.RestoreDirectory = true ;

    if(openFileDialog1.ShowDialog() != DialogResult.OK)
        return;
    FileStream fs = new FileStream(openFileDialog1.FileName, FileMode.Open);
    BinaryReader r = new BinaryReader(fs);
    if (r.ReadString() != "Fichier Employé")
        MessageBox.Show("Fichier sélectionné invalide");
    else
    {
        m_Compteur = r.ReadInt32();
        for (int i=0 ; i<m_Compteur; i++)
        {
            m_TNom[i] = r.ReadString();
            m_TAge[i] = r.ReadInt32();
            m_TSalaire[i] =r.ReadDouble();
        }
    }
    r.Close();
    fs.Close();
}
```

3. Examiner et Comprendre les méthodes:

btnEnregistrer_click() et btnRechercher_click()

4. Exécuter l'application et entrer 10 éléments suivants dans les tableaux

Jacques	32	25334.89
Jean-Paul	43	40434.77
Sylvie	25	52677.84
Sophie	56	22344.98
Patrick	47	56888.55
Josée	22	25000.11
Gaston	59	23777.77
Lise	70	66666.66
Daniel	44	55555.55
Louise	66	57999.99

5. Enregistrer dans un fichier nommé: donnees.emp

6. Ajouter un bouton pour faire afficher dans un MessageBox:

Le Salaire le plus élevé est: xxxxx.xx \$

7. Ajouter un bouton pour faire afficher dans un MessageBox:

Le Salaire le plus bas est: xxxxx.xx \$

8. Ajouter un bouton pour faire afficher dans un MessageBox:

La moyenne d'âge est: xx.xx

9. Ajouter un bouton pour faire afficher dans un seul MessageBox:

Le nom, l'âge et le salaire de chaque employé dans la forme suivante:

Nom: xxxxxxxxxxxx Âge: xx Salaire: xxxxxx.xx \$

Nom: xxxxxxxxxxxx Âge: xx Salaire: xxxxxx.xx \$

10. Ajouter un bouton pour faire afficher dans un MessageBox:

La personne la plus âgée est: xxxxxxxxxxxx Son âge est: xx Son salaire: xxxxxx.xx \$

12.8 Travail Pratique No. 8 : La structure répétitive

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes;
- concevoir une application en Visual C#;
- respecter les normes de programmation;
- pratiquer l'utilisation des structures séquentielles, alternatives et répétitives;
- implanter la validation;
- utiliser les notions de traduction en langage C#;
- utiliser les types de données;
- expérimenter la lecture, la compréhension et la complétude de code déjà écrit.

Date de remise:

Point:

Correction:

Directives: Compléter l'application de base fournie par le prof, en écrivant la méthode Essai.

Vous disposez de l'application TP8 PENDU, sur le serveur, développée en C#.

La conception visuelle de l'application est déjà faite de même que la méthode du menu pour commencer une nouvelle partie, la méthode Dessine qui dessine une portion du bonhomme si le joueur essaye une lettre qui n'est pas dans le mot ainsi que la méthode Paint() qui rafraîchit l'affichage du dessin lorsque nécessaire.

L'application dispose d'un tableau de mots à trouver. La méthode Nouvelle partie sélectionne un mot que l'utilisateur devra trouver en essayant des lettres. Cette méthode affiche dans la boîte d'édition Mot à trouver, les lettres du mot recherché remplacées par des ?. Elle affiche aussi le nombre de lettres composant le mot.

Il s'agit pour vous de programmer la méthode **Essai** pour poursuivre le jeu.

L'utilisateur choisit une lettre. La méthode Essai doit :

- Valider le caractère saisi:

S'il y a bien eu un caractère entré.

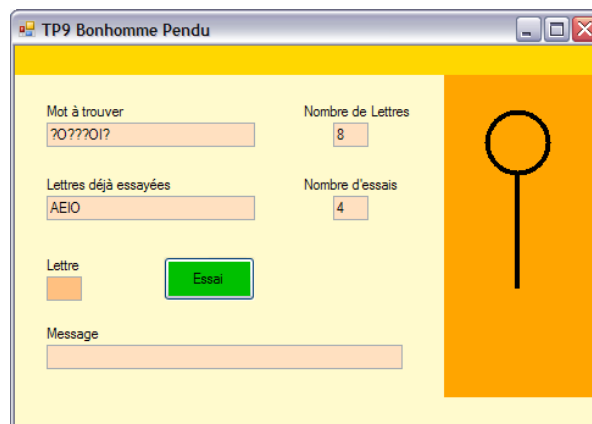
Si le caractère correspond bien à une lettre (A à Z ou le trait d'union)

Si la lettre n'a pas déjà été essayée.

Dans chaque cas, vous devez afficher un message d'erreur approprié dans la boîte de saisie Message.

Votre méthode doit aussi bien accepter les majuscules que les minuscules.

- Ajouter la lettre saisie aux lettres déjà essayées.
- Pour chaque lettre correspondante du mot à trouver remplacer le point d'interrogation par la lettre.
- Augmenter le nombre d'essai de 1.
- Si le mot entier a été découvert, afficher le message BRAVO et désactiver le bouton essai et la zone d'édition Lettre. Activer le bouton Nouvelle.
- Dans le cas où la lettre n'est pas présente, augmenter la variable m_Dessin de 1 et appeler la méthode Dessine.
- Si les 10 parties du corps du bonhomme ont été dessinées, vous devez afficher un message indiquant que le joueur a perdu ainsi que la solution et terminer la partie.



```
// Variables membres
int m_Dessin;
int m_NombreEssais;
string m_MotCherche;
string m_MotATrouver;
System.Drawing.Graphics m_FormsGraphics;
```

```
// Méthode qui initialise une nouvelle partie
private void mnuNouvellePartie_Click(object sender, EventArgs e)
{
    string[] ListeMots = new string[10] {"AUTOMNE", "COLLEGE", "ORDINATEUR", "FICHER",
        "LANGAGE", "SESSION", "ECONOMIE", "SOUSBOIS", "DIMANCHE", "JANVIER"};
    int indice;
```

```

        int NombreDeLettre;
        System.Random GenerateurAleatoire = new System.Random();

        m_MotCherche = ListeMots[GenerateurAleatoire.Next(0, 10)];
        NombreDeLettre = m_MotCherche.Length;
        m_MotATrouver = "";
        for (indice = 0; indice < NombreDeLettre; indice++)
            m_MotATrouver += '?';
        m_Dessin = 0;
        m_NombreEssais = 0;

        txtMotATrouver.Text = m_MotATrouver;
        txtNombreLettres.Text = NombreDeLettre.ToString();
        txtLettre.Text = "";
        txtMessage.Text = "";
        txtLettresDejaEssayees.Text = "";
        txtNombreEssais.Text = m_NombreEssais.ToString();
        panJeu.Visible = true;
        panJeu.Enabled = true;
        panDessin.Invalidate();
        mnuNouvellePartie.Visible = false;
        txtLettre.Enabled = true;
        btnEssai.Enabled = true;
        txtLettre.Focus();
    }
}
//Méthode qui dessine une partie du bonhomme selon la variable m_Dessin
public void Dessine()
{
    SolidBrush MaBrosse = new SolidBrush(Color.Black);
    Pen MonCrayon = new Pen(MaBrosse, 4);

    int[,] TCoord =
    {{35, 30, 50, 50},
     {60, 80, 60, 175},
     {60, 175, 40, 225},
     {60, 175, 80, 225},
     {60, 130, 20, 120},
     {60, 130, 100, 120},
     {45, 40, 9, 9},
     {65, 40, 9, 9},
     {56, 55, 7, 7},
     {50, 70, 70, 70}};

    if (m_Dessin == 1 || m_Dessin >= 7 && m_Dessin <= 9)
        m_FormsGraphics.DrawEllipse(MonCrayon, TCoord[m_Dessin, 0], TCoord[m_Dessin, 1],
                                     TCoord[m_Dessin, 2], TCoord[m_Dessin, 3]);
    else
        m_FormsGraphics.DrawLine(MonCrayon, TCoord[m_Dessin, 0], TCoord[m_Dessin, 1],
                                  TCoord[m_Dessin, 2], TCoord[m_Dessin, 3]);

    MaBrosse.Dispose();
    MonCrayon.Dispose();

    // ré-Affichage du dessin lors du rafraîchissement de la formulaire par Windows
private void frmTP8_Pendu_Paint(object sender, PaintEventArgs e)
{
    int Temp = m_Dessin;
    for (m_Dessin = 0; m_Dessin < Temp; m_Dessin++)
        Dessine();
    m_Dessin = Temp;
}

```

12.9 Travail Pratique No. 9 : Les Boucles imbriquées et les méthodes de dessin

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- Consolider la théorie sur les boucles;
- Utiliser des boucles imbriquées dans un programme;
- Manipuler des méthodes graphiques.

Date de remise:

Point:

Correction:

Directives:

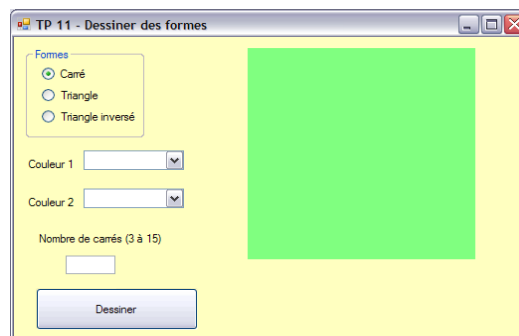
L'exécutable du programme est disponible sur le serveur. À consulter pour la conception visuelle et des exemples d'exécution avant de programmer votre application.

Concevoir une application qui dessinera de petits carrés de 10 pixels X 10 pixels selon trois formes déterminées.

La couleur des petits carrés alternera selon 2 couleurs choisies par l'utilisateur.

L'utilisateur aura le choix de former 3 figures géométriques: Carré, Triangle ou Triangle inversé.

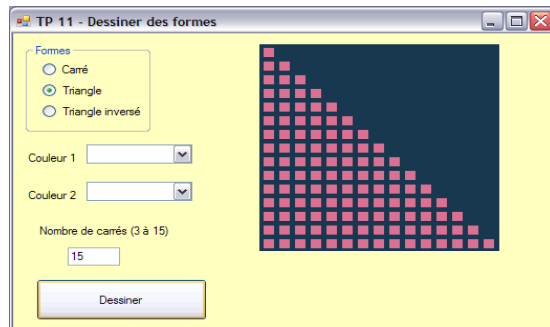
La dimension des figures peuvent être de 3 à 15 petits carrés.



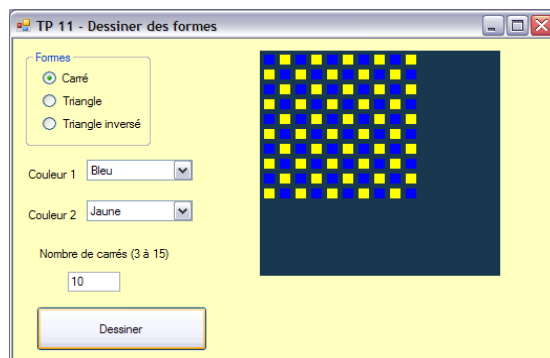
- La forme des figures doit être choisie à l'aide de boutons radio.
- Les deux couleurs sont choisies dans des ComboBox. (Si aucun choix, prévoir 2 couleurs par défaut).
- Le nombre de carrés doit être validé.

- Le dessin se fera sur un contrôle Panel.

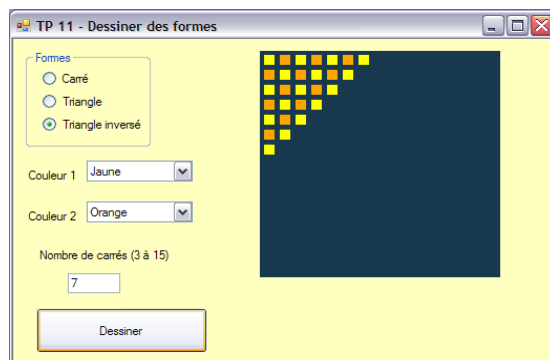
Exemple 1



Exemple 2



Exemple 3



12.10 Travail pratique No. 10 : Les compteurs et accumulateurs.

Objectifs spécifiques:

La maîtrise de ces objectifs sera évaluée aux examens.

- expérimenter les étapes de résolution de problèmes;
- concevoir une application en Visual C#;
- respecter les normes de programmation;
- pratiquer l'utilisation des structures séquentielles et des structures alternatives;
- distinguer les niveaux de traitements (méthode de traitement détail et méthode de traitement final);
- utiliser les boutons de radio, les cases à cocher et les ComboBox;
- utiliser les notions de traduction en langage C#;
- utiliser les types de données;
- concevoir les données de test afin de valider les résultats de l'application.

Date de remise:

Point:

Correction:

Directives: Faire l'analyse, la conception visuelle, l'algorithme, les jeux d'essais et la trace avant le début du prochain laboratoire. L'élaboration de l'application sur ordinateur se fera au prochain laboratoire. Votre travail sur papier sera vérifié.

Énoncé : L'agence de voyage Beau Soleil veut avoir une application en C# lui permettant de **calculer et d'afficher rapidement le prix réel pour un voyage, le prix total pour le groupe et la commission** qu'elle donnera aux vendeurs lorsqu'ils effectuent la vente d'un voyage de groupes offert en tenant compte des critères suivants:

- La destination (Floride, Mexique ou Espagne)
- La saison (Basse Saison ou Haute Saison)
- La quantité vendue (le nombre de personnes composant le groupe)
- La période de départ (Sur Semaine ou Fin de Semaine)
- Le prix de base du voyage fixé par l'agence
- Les options du voyage : Accompagnateur, Chambre de luxe, Petits déjeuners inclus.

Chaque option sélectionnée **augmente** le prix de base du voyage saisi :

- + 3 % du prix du voyage si on a choisi un accompagnateur;
- + 5 % du prix du voyage si on a choisi le chambre de luxe;
- + 7 % du prix du voyage si on a choisi les petits déjeuners inclus.

NOTE: Tous les montants de bonus et de pénalité se calculent sur le prix réel d'un voyage (et non pas sur le prix global pour le groupe).

La commission se calcule de la façon suivante:

- Pour la destination

Floride: Commission de base: 50.00 \$
plus un bonus de 2 % si le départ est sur semaine.

Mexique: Commission de base: 75.00 \$
moins une pénalité de 1 % si le départ est la fin de semaine.

Espagne: Commission de base: 100.00 \$
plus un bonus de 3% si le départ est sur semaine
plus un bonus de 2% si le départ est en basse saison.

- Pour toutes les destinations on obtient un bonus supplémentaire de:

2% si le **prix global pour le groupe** est entre 1000 \$ inclus et 2000 \$ exclus

3 % si le **prix global pour le groupe** est entre 2000 \$ et 5000 \$ exclus

5% si le **prix global pour le groupe** est de 5000 \$ et plus.

Vous devez afficher le prix réel d'un voyage ainsi que le prix global pour le groupe et la commission donnée au vendeur.

À la fin du traitement seulement, faire afficher le total et la moyenne des commissions accordées.
Jeux d'essais

Remettre un formulaire par essai ainsi que le formulaire affiché à la fin du traitement.

Destination	Saison	Période de départ	Prix de base du voyage	Quantité	Options
Floride	Haute	Semaine	450	2	A
Floride	Basse	Fin de semaine	500	2	A,CL
Mexique	Basse	Semaine	500	3	A,CL,PD
Mexique	Haute	Fin de semaine	500	4	CL
Espagne	Haute	Semaine	500	9	CL,PD
Espagne	Basse	Semaine	600	10	PD
Espagne	Basse	Fin de semaine	750	25	aucune
Mexique	Haute	Semaine	500	10	aucune

13 Annexe 1 – Table des codes ASCII

Ctrl	Déc	Hex	Car	Code	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car
^@	0	00		NUL	32	20	!	64	40	@	96	60	'
^A	1	01		SOH	33	21	!	65	41	A	97	61	a
^B	2	02		STX	34	22	..	66	42	B	98	62	b
^C	3	03		ETX	35	23	#	67	43	C	99	63	c
^D	4	04		EOT	36	24	\$	68	44	D	100	64	d
^E	5	05		ENQ	37	25	%	69	45	E	101	65	e
^F	6	06		ACK	38	26	&	70	46	F	102	66	f
^G	7	07		BEL	39	27	,	71	47	G	103	67	g
^H	8	08		BS	40	28	(72	48	H	104	68	h
^I	9	09		HT	41	29)	73	49	I	105	69	i
^J	10	0A		LF	42	2A	*	74	4A	J	106	6A	j
^K	11	0B		VT	43	2B	+	75	4B	K	107	6B	k
^L	12	0C		FF	44	2C	,	76	4C	L	108	6C	l
^M	13	0D		CR	45	2D	-	77	4D	M	109	6D	m
^N	14	0E		SO	46	2E	.	78	4E	N	110	6E	n
^O	15	0F		SI	47	2F	/	79	4F	O	111	6F	o
^P	16	10		DLE	48	30	0	80	50	P	112	70	p
^Q	17	11		DC1	49	31	1	81	51	Q	113	71	q
^R	18	12		DC2	50	32	2	82	52	R	114	72	r
^S	19	13		DC3	51	33	3	83	53	S	115	73	s
^T	20	14		DC4	52	34	4	84	54	T	116	74	t
^U	21	15		NAK	53	35	5	85	55	U	117	75	u
^V	22	16		SYN	54	36	6	86	56	V	118	76	v
^W	23	17		ETB	55	37	7	87	57	W	119	77	w
^X	24	18		CAN	56	38	8	88	58	X	120	78	x
^Y	25	19		EM	57	39	9	89	59	Y	121	79	y
^Z	26	1A		SUB	58	3A	:	90	5A	Z	122	7A	z
^[27	1B		ESC	59	3B	;	91	5B	[123	7B	{
^\	28	1C		FS	60	3C	<	92	5C	\	124	7C	
^]	29	1D		GS	61	3D	=	93	5D]	125	7D	}
^^	30	1E	▲	RS	62	3E	>	94	5E	^	126	7E	~
^-	31	1F	▼	US	63	3F	?	95	5F	_	127	7F	Δ*

* Le code ASCII 127 correspond au code de suppression. Avec MS-DOS, ce code produit le même effet que ASCII 8 (BS). Le code de suppression peut être généré par la combinaison de touches CTRL + RET. ARR.

Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car	Déc	Hex	Car
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ü	161	A1	í	193	C1	┐	225	E1	β
130	82	é	162	A2	ó	194	C2	┌	226	E2	Γ
131	83	â	163	A3	ú	195	C3	└	227	E3	Π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	+	229	E5	σ
134	86	â	166	A6	à	198	C6	†	230	E6	μ
135	87	ç	167	A7	ø	199	C7	‡	231	E7	Υ
136	88	ê	168	A8	¿	200	C8	‖	232	E8	ϕ
137	89	ë	169	A9	ˆ	201	C9	ƒ	233	E9	Θ
138	8A	è	170	AA	˜	202	CA	≡	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	⌈	235	EB	δ
140	8C	î	172	AC	¼	204	CC	└	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	=	237	ED	Φ
142	8E	Ä	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Å	175	AF	»	207	CF	±	239	EF	Π
144	90	É	176	B0	⋄	208	D0	⌋	240	F0	≡
145	91	æ	177	B1	▮	209	D1	⌈	241	F1	±
146	92	Ē	178	B2	▮	210	D2	┌	242	F2	≥
147	93	ô	179	B3	—	211	D3	┌	243	F3	≤
148	94	ö	180	B4	└	212	D4	└	244	F4	┌
149	95	ò	181	B5	└	213	D5	ƒ	245	F5	└
150	96	û	182	B6	└	214	D6	└	246	F6	÷
151	97	ù	183	B7	└	215	D7	‡	247	F7	≈
152	98	ÿ	184	B8	└	216	D8	‡	248	F8	◊
153	99	ö	185	B9	└	217	D9	└	249	F9	•
154	9A	Ü	186	BA	└	218	DA	└	250	FA	·
155	9B	ç	187	BB	└	219	DB	▮	251	FB	√
156	9C	£	188	BC	└	220	DC	▮	252	FC	ⁿ
157	9D	¥	189	BD	└	221	DD	▮	253	FD	²
158	9E	℥	190	BE	└	222	DE	▮	254	FE	■
159	9F	ƒ	191	BF	└	223	DF	▮	255	FF	