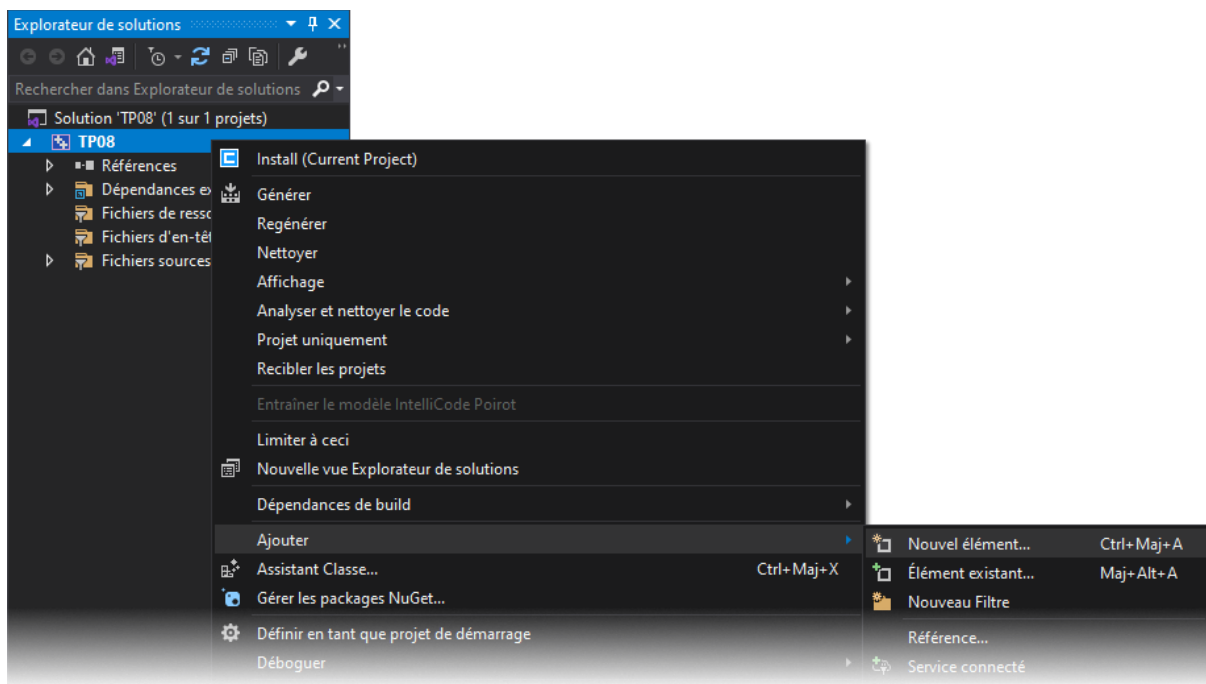


TP 08 – Fonctions

Exercice 1 Mise en bouche

1.1 Calculs mathématiques

- Créez une nouvelle solution Visual Studio comme vous en avez l'habitude, puis ajoutez-y un couple de fichiers *maths.h* / *maths.cpp*. L'ajout de fichiers sources se fait par un clic droit sur le projet puis *Ajouter / Nouvel élément...* comme sur la capture ci-dessous.



- Dans ce binôme de fichiers, créez une fonction qui calcule selon les paramètres nombres réels **moyenne** (μ), **sigma** (σ) et **x** la valeur

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- Pour calculer e^{arg} , vous pouvez utiliser la fonction `std::exp(double arg)` disponible dans la bibliothèque `<cmath>`
 - Pour la valeur de π , vous pouvez utiliser la constante `std::numbers::pi` définie dans la bibliothèque `<numbers>` si votre compilateur supporte C++20, ou bien définir votre propre constante à 3.1415926535897932384626433832795.
- Dans un programme principal, vous testerez cette fonction selon le tableau suivant :

Tableau 1 : valeurs de test pour la fonction $f(x)$

x	μ	σ	$f(x)$
0	0	1	0.398942
1	0	1	0.241971
0	1	1	0.241971
1	0	2	0.176033

1.2 Calculs mathématiques en lot

- 📁 Créez une nouvelle fonction qui calculera tous les $f(x)$ pour des valeurs x régulièrement réparties entre deux bornes. En plus de ces deux bornes, vous spécifierez en paramètres de la fonction les valeurs de μ , σ et le nombre d'échantillons entre les deux bornes. La fonction retournera toutes ces valeurs stockées dans un tableau de réel.
- 📁 Dans un programme principal, vous testerez cette fonction en vous assurant d'obtenir les 5 valeurs suivantes pour une répartition entre $x = -4$ et $x = 4$ avec $\mu = 0$ et $\sigma = 1$:

Tableau 2 : valeurs de test de $f(x)$ pour une répartition régulière de 5 valeurs entre $x = -4$ et $x = 4$ avec $\mu = 0$ et $\sigma = 1$

x	-4	-2	0	2	4
$f(x)$	0.00013383	0.053991	0.398942	0.053991	0.00013383

1.3 Affichage dans la console

- 📁 Créez une fonction d'affichage dans un nouveau couple de fichiers `console.h` / `console.cpp` qui affichera l'allure de la courbe $f(x)$ à la manière de ce qui avait été fait à l'exercice 3.2 du TP 04 – Tableaux :

Nous allons considérer les valeurs numériques contenues dans les cases du tableau comme des hauteurs. Pour chaque case, nous afficherons une pile de 'o' dont la hauteur sera proportionnelle à la valeur de la case. La hauteur maximale sera de $\frac{N}{4}$ 'o'. Vous pouvez donc calculer pour chaque case sa hauteur en nombre de 'o'.

Citation 1 : Extrait du TP 04 – Tableaux.

- 📁 Pour cette fois-ci, la hauteur maximale sera un paramètre de la fonction d'affichage.

📁 Si vous souhaitez utiliser une variable `max` et que vous avez inclus `<Windows.h>`, vous aurez une erreur de compilation due à un choix de Microsoft au début des années 90 qui ne s'avère plus pertinent aujourd'hui. Pour éviter ce problème, vous devez ajouter l'instruction **`#define NOMINMAX`** avant d'inclure `<Windows.h>`.

- 📁 Dans un programme principal, vous testerez cette fonction en affichant l'allure de la courbe pour 200 valeurs de x comprises entre -4 et 4 avec $\mu = 0$ et $\sigma = 1$. Fixez la hauteur maximale de l'affichage à 50.
- 📁 L'affichage devrait vous rappeler des souvenirs.

1.4 Programme principal – facultatif

- 📁 Créez un programme principal basé sur un menu qui permet de régler les différents paramètres du calcul et de l'affichage de la courbe.

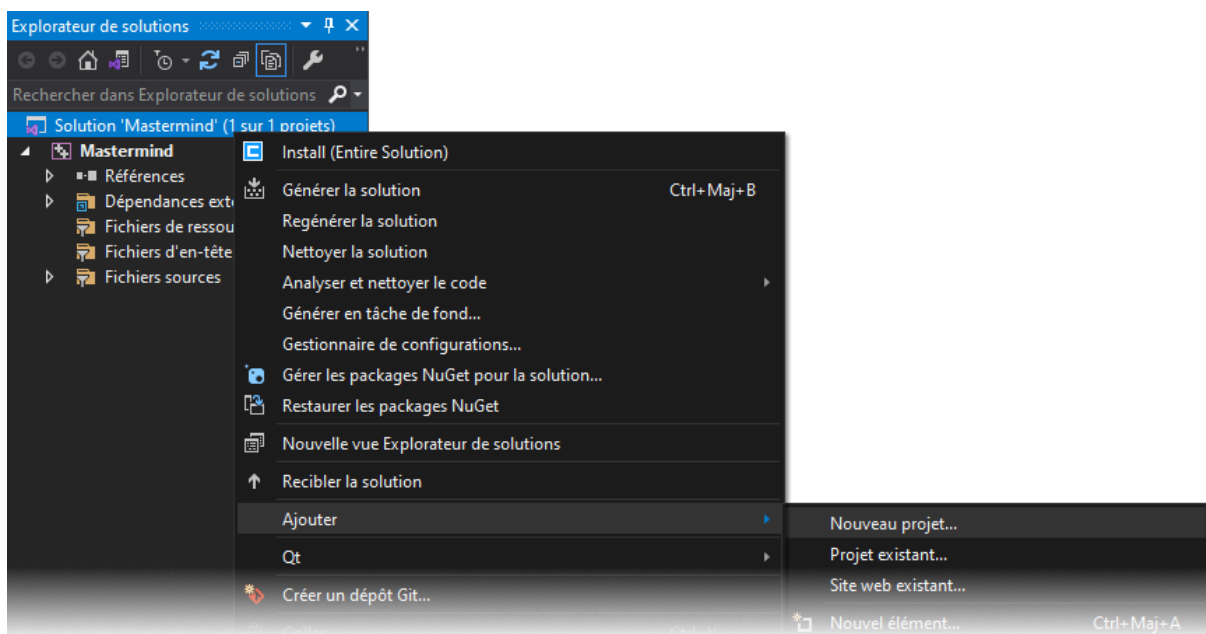
Exercice 2 Mastermind

Le Mastermind est un jeu de société dont le but est de trouver un code caché par déductions successives. Les règles du jeu sont exposées sur [Wikipédia](https://fr.wikipedia.org/wiki/Mastermind). L'objectif de cet exercice sera d'implémenter totalement ce jeu avec un système de meilleurs scores enregistrés. Nous allons procéder étape par étape.

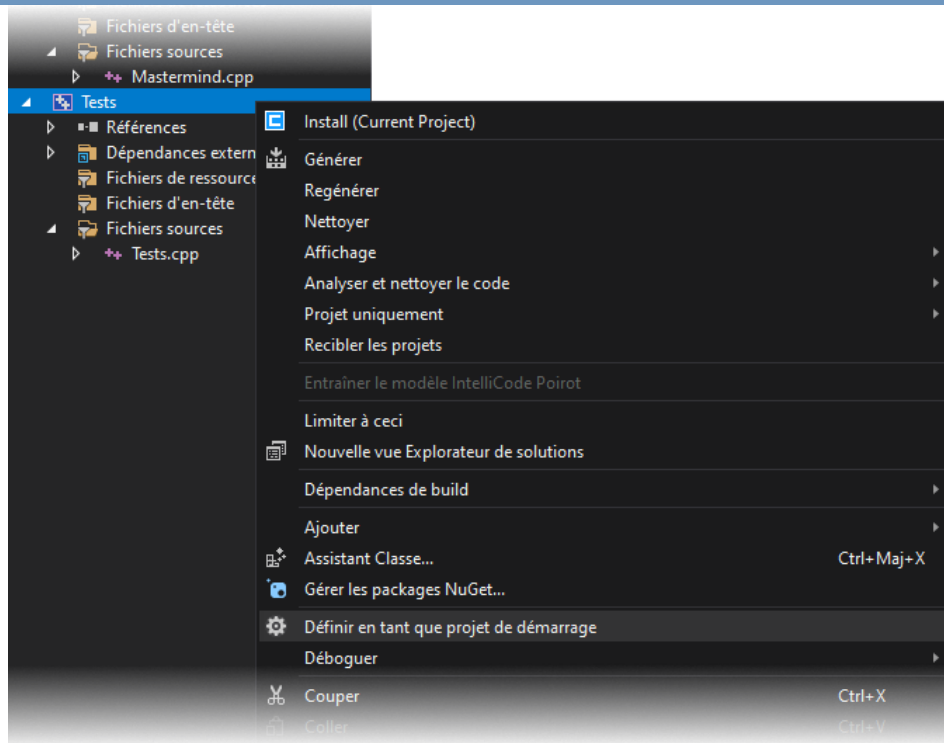
Pour vous satisfaire et vous montrer l'objectif, une version compilée du résultat attendu est disponible dans le fichier *VEtudiant.zip* disponible sur le cours en ligne.

2.1 Mise en place de la solution Visual Studio

- 📁 Créez une nouvelle solution Visual Studio comme vous en avez l'habitude et que vous nommerez *Mastermind*.
- 🔗 La solution contient un unique projet du même nom.
- 📁 Par le biais d'un clic droit sur la solution *Mastermind* dans l'explorateur de solution, ajoutez un nouveau projet de type *Application console* à votre solution.



- 🔗 Nommez ce nouveau projet « *Tests* »
- 🔗 Définissez ce nouveau projet comme projet de démarrage en faisant un clic droit sur celui-ci et en choisissant « *Définir en tant que projet de démarrage* »



- Comme pour tout nouveau projet, définissez la norme du langage et supprimez la compilation en 32 bits.

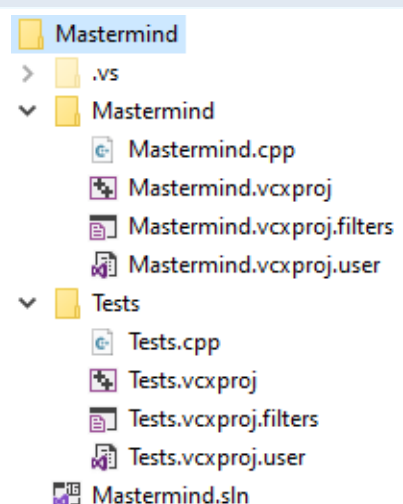
Le projet *Mastermind* contiendra le code du jeu complet. Le projet *Tests* sera seulement utile pour tester séparément l'ensemble des fonctionnalités du jeu, sans avoir à dérouler une partie. Nous commencerons donc à programmer dans le projet *Tests*.

Pour ne pas vous tromper, vous devez **SCRUPULEUSEMENT** et **RIGOREUSEMENT** suivre les instructions.

Si vous avez suivi l'ensemble de ces étapes correctement, vous devez avoir une hiérarchie de fichier sur votre disque dur comme sur la figure ci-contre.

Le répertoire *Mastermind\Mastermind* contiendra tout le code source du projet complet. C'est dans ce répertoire que devront se trouver tous les fichiers sources à ajouter.

Le répertoire *Mastermind\Tests* ne contiendra que le fichier *Tests.cpp* qui sera un programme permettant de tester séparément les fonctionnalités du jeu de Mastermind. Le projet *Tests* fera référence aux autres fichiers sources contenus dans le répertoire principal *Mastermind\Mastermind*.



2.2 Génération d'un code aléatoire

- Dans le projet *Mastermind*, renommez le fichier *Mastermind.cpp* en *main.cpp*. Ce fichier ne contiendra que la fonction `main()`.
- Toujours dans le projet *Mastermind*, ajoutez un couple de fichiers *mastermind.h* / *mastermind.cpp*.

Dans le fichier *mastermind.h*, vous allez définir un ensemble de constantes et d'alias de type pour rendre votre programme plus robuste aux potentiels changements futurs et plus lisible.

- Définissez une constante contenant le nombre de couleurs de pion du Mastermind. La valeur typique est de 6 couleurs.
- Définissez une constante contenant la taille du code en nombre de pions. La valeur typique est de 4 pions.

On peut choisir de coder les codes du Mastermind par un tableau fixe d'entiers, chaque entier codant pour une couleur. La taille du tableau est donc la taille d'un code Mastermind. Le type de ce tableau sera donc quelque chose du style `std::array<size_t, codeSize>`. Utiliser ce type à chaque fois que nous manipulerons un code Mastermind dans nos sources va vite devenir pénible. Le C++ nous permet de définir des alias de type, c'est-à-dire de renommer plus simplement un type compliqué. Cela se fait par l'instruction `using` qui s'utilise selon cette syntaxe :

```
using nouveau_nom = type_complique;
```

Ainsi, à chaque fois que le compilateur verra « *nouveau_nom* », il comprendra « *type_complique* ».

- Définissez dans le fichier *mastermind.h* un alias nommé `code_t` du type des codes du Mastermind.
- Toujours dans le couple de fichiers *mastermind.h* / *mastermind.cpp*, créez une fonction `generateCode()` qui retournera un nouveau code Mastermind aléatoire.
 - Pour rappel, la génération de nombres aléatoires a été abordée dans l'exercice 5 du TP 04 – Tableaux.

Nous allons maintenant devoir tester cette fonction pour nous assurer de son bon fonctionnement.

- Dans le projet *Tests*, ajouter les fichiers *mastermind.h* et *mastermind.cpp* du projet Mastermind. Pour cela, au lieu de sélectionner « *Nouvel élément...* » dans le menu d'ajout, choisissez « *Élément existant...* » puis sélectionner les fichiers dans le répertoire du projet Mastermind.
- Dans le fichier *Tests.cpp* du projet *Tests*, incluez le fichier *mastermind.h*. Comme il ne se trouve pas dans le même répertoire que *Tests.cpp*, vous devez spécifier son chemin d'accès relatif. Si vous avez suivi scrupuleusement les instructions jusqu'ici, la ligne d'inclusion est :

```
#include "../Mastermind/mastermind.h"
```

- Dans le fichier *Tests.cpp*, créez un petit programme appelant 4 fois la fonction `generateCode()` et affichant les 4 codes ainsi générés.
 - Vous devriez avoir une sortie du type de celle de la figure suivante :

```
4 codes aléatoires :
0 4 4 2
3 0 5 4
2 3 5 2
0 4 2 3
```

2.3 Affichage des codes

Les codes affichés sous forme de chiffres ne sont pas dans l'esprit du Mastermind qui propose des pions de couleur. Nous allons convertir les codes numériques du type `code_t` en chaîne de caractères dont chaque chiffre sera affiché avec une lettre représentant une couleur. Cela sera donc une problématique d'affichage et ne doit donc pas être mélangé avec les fonctions de logique du Mastermind.

📁 Ajoutez **au projet Mastermind** un nouveau binôme de fichiers `mastermind_cui.h` / `mastermind_cui.cpp`. L'acronyme « *cui* » a été choisi pour « *Console User Interface* ».

📁 Créez dans ce binôme une fonction `toString()` prenant en paramètre un `code_t` et qui retourne un `std::string`.

🖨 Le tableau suivant vous propose un exemple de retranscription chiffre → caractère de couleur :

Chiffre du code	0	1	2	3	4	5
Caractère de couleur	R	G	B	Y	W	K
Couleur	Rouge	Vert	Bleu	Jaune	Blanc	Noir

📁 Dans le projet Tests, testez cette fonction en affichant les codes générés aléatoirement sous forme de texte.

🖨 Vous devriez avoir une sortie du type de celle de la figure suivante :

```
4 codes aléatoires :
5 3 4 3 <=> K Y W Y
5 1 0 0 <=> K G R R
0 1 2 2 <=> R G B B
2 4 2 3 <=> B W B Y
```

2.4 Vérification d'un code

Dans les règles du Mastermind, lorsqu'un nouveau code est proposé, le joueur reçoit des indices sous forme de petits pions noir ou blanc. Il reçoit un pion noir pour chaque pion de couleur qu'il a bien placé dans le code puis un pion blanc pour chaque pion de couleur restant qui correspond à une bonne couleur dans le code mais à la mauvaise place.

Voici quelques exemples de résultat selon le code à trouver et le code proposé :

Code à trouver	Code proposé	Résultat des indices
R G B B	B B B B	●●
R G B B	B W Y R	○○
R G B B	B B G R	○○○○
R G B B	R B B G	●●○○
R G B B	R G B B	●●●●

On peut donc représenter les indices comme étant un tableau de 0 à 4 booléens. Une valeur **true** pouvant signifier qu'un pion est bien placé (petit pion noir) et une valeur **false** qu'un pion est de la bonne couleur mais mal placé (petit pion blanc).

- 📁 Dans les fichiers *mastermind.h* / *mastermind.cpp*, créez une fonction **checkCode()** qui prendra en paramètre le code à trouver et le code proposé. Elle retournera un tableau de booléen correspondant aux indices que le joueur recevra.
 - ➡ A l'instar de l'alias **code_t**, vous pouvez définir un alias **check_t**.
 - ➡ Vous devez tester dans un premier temps les pions bien placés et vous assurer qu'ils ne seront plus utilisés pour la vérification des pions mal placés. Pour ce faire, vous pouvez changer leurs chiffres à des valeurs qui ne correspondent pas à des couleurs.
- 📁 Afin de tester votre fonction, ajouter au programme *Tests* les tests proposés en exemple dans le tableau précédent.
 - ➡ Vous devriez avoir une sortie du type de celle de la figure suivante :

Tests de la vérification de code :		
Code à trouver	code proposé	indices
R G B B	B B B B	11
R G B B	B W Y R	00
R G B B	B B G R	0000
R G B B	R B B G	1100
R G B B	R G B B	1111

2.5 Affichage des indices de découverte d'un code

Comme pour l'affichage des codes, l'affichage des indices sous forme de 0 ou 1 n'est pas du plus bel effet.

- 📁 A l'instar de l'exercice 2.3, créez une nouvelle fonction **toString()** prenant en paramètre un **check_t** et retournant la chaîne de caractères correspondante. Vous pourrez coder les petits pions noirs avec le caractère « • » et les petits pions blancs avec le caractère « o ».
- ➡ Le C++ accepte que deux fonctions portent le même nom (ici **toString**) à condition qu'elles n'aient pas les mêmes paramètres.
- 📁 Testez cette nouvelle fonction en affichant les indices précédents également sous forme de chaîne de caractères.
 - ➡ Vous devriez avoir une sortie du type de celle de la figure suivante :

Tests de la vérification de code :			
Code à trouver	code proposé	indices	
R G B B	B B B B	11	●●
R G B B	B W Y R	00	oo
R G B B	B B G R	0000	oooo
R G B B	R B B G	1100	●●oo
R G B B	R G B B	1111	●●●●

2.6 Saisie d'un code par l'utilisateur

Lorsque l'utilisateur voudra tester un code, il devra bien sûr le saisir au clavier.

- Dans le binôme de fichiers adéquat, créez une fonction **askForCode()** qui retournera un tableau de 4 caractères correspondant aux quatre couleurs du code saisi.
 - ➡ La fonction devra ignorer les espaces
 - ➡ La fonction devra ignorer et alerter l'utilisateur en cas de saisie d'un caractère qui ne fait pas partie du tableau des couleurs.
- Ajouter le test de cette fonction dans le programme *Tests*.
 - ➡ Vous devriez avoir une sortie du type de celle de la figure suivante :

```
Test de la fonction askForCode() :  
Votre nouveau code :  
r G B Y  
Le code couleur 'r' n'est pas reconnu. Il est ignoré.  
K  
Résultat de la fonction askForCode() : GBYK
```

Dans cette figure, la sortie console de la fonction `askForCode()` est sur fond bleu, les caractères orange sont saisis par l'utilisateur.

2.7 Conversion d'un code saisi par l'utilisateur en code Mastermind

Le tableau de caractère saisi par l'utilisateur et retournée par la fonction `askForCode()` ne peut pas être exploité par les fonctions de logique du Mastermind. Il doit d'abord être converti en `code_t`.

- Dans le binôme de fichiers adéquat, créez une fonction **fromChars()** qui convertira un code retourné par **askForCode()** en un code de type **code_t**.
- Testez cette fonction dans le programme *Tests*.
 - ➡ Vous devriez avoir une sortie du type de celle de la figure suivante :

```
Test de la fonction askForCode() :  
Votre nouveau code :  
r G B Y  
Le code couleur 'r' n'est pas reconnu. Il est ignoré.  
K  
Résultat de la fonction askForCode() : GBYK  
Une fois converti en code_t puis affiché en chaine, ce code est G B Y K
```


2.8 Affichage du plateau de jeu

Nous allons représenter le plateau du jeu comme sur la figure ci-dessous :




```

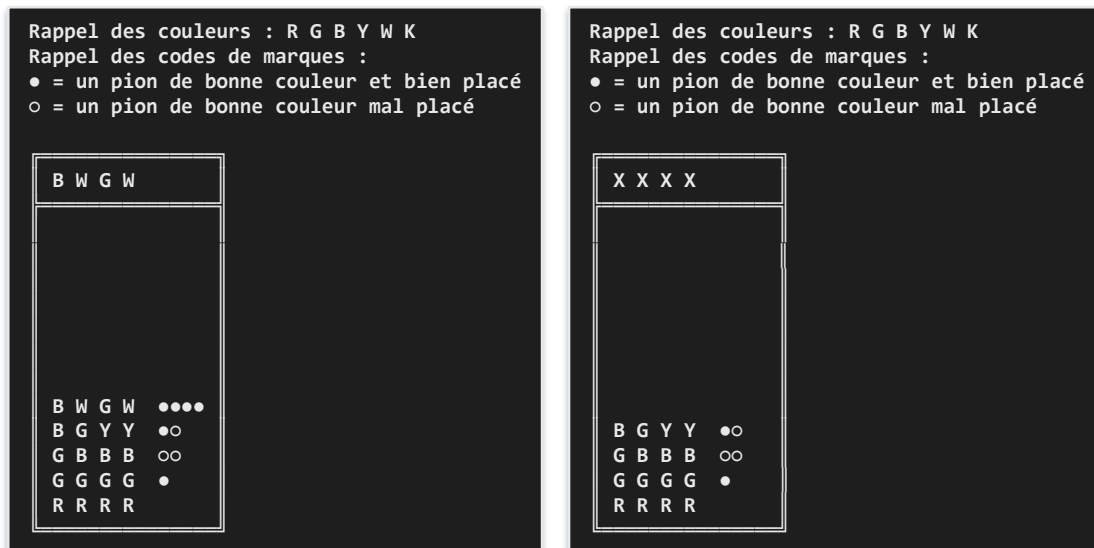
01: Rappel des couleurs : R G B Y W K
02: Rappel des codes de marques :
03: ● = un pion de bonne couleur et bien placé
04: ○ = un pion de bonne couleur mal placé
05:
06:
07: B W G W
08:
09:
10:
11:
12:
13:
14:
15:
16: B W G W ●●●●
17: B G Y Y ●○
18: G B B B ○○
19: G G G G ●
20: R R R R
21:

```

Cette figure est constituée de la façon suivante :



1. Le rappel des différents codes est affiché
2. Une première « cellule » sur les lignes 6 à 8 du code à trouver est affichée. Si le code n'a pas encore été déterminé, alors le code est « masqué » par des « X ».
3. Les lignes 9 à 20 correspondent à l'affichage des 12 tentatives de détermination du code à trouver. Les tentatives sont affichées du bas vers le haut, c'est-à-dire que la ligne 20 correspond à la première tentative, la ligne 9 à la douzième. Une ligne de tentative est constituée d'un filet vertical double « || » puis d'un espace, puis de l'affichage du code de la tentative sur 7 caractères, puis de deux espaces, puis de l'affichage la vérification du code sur 4 caractères maximum et enfin d'un espace suivi d'un nouveau file vertical double « || ».

-  Créez une fonction **showGame()** prenant en paramètre le code à trouver et un tableau de codes tentés. Elle affichera dans la console le plateau de jeu comme présenté ci-dessus.
-  Testez votre fonction dans le programme *Tests* en lui fournissant les paramètres correspondant à la figure ci-dessus et une nouvelle fois en supprimant la dernière tentative.
-  Vous devriez avoir une sortie du type de celle de la figure suivante :




2.9 Jouons un peu

Nous avons à présent tout ce qu'il faut pour jouer, il ne reste « plus qu'à » écrire le dérouler d'une partie.


-  Dans une fonction **runGame()**, écrivez l'algorithme d'une partie :
 - ➡ Générez un nouveau code aléatoire.
 - ➡ Vous devez ensuite, en boucle, réaliser les étapes suivantes
 - ➡ Afficher le plateau de jeu (avant d'afficher le plateau, il est recommandé d'effacer le contenu de la console en appelant la fonction **std::system("cls");**)
 - ➡ Demander un code
 - ➡ Ajouter ce code au tableau des tentatives
 - ➡ La boucle précédente doit s'arrêter quand le nombre de tentatives est arrivé à 12 ou si le code de la dernière tentative est le code recherché.
 - ➡ En fonction du critère qui a arrêté la boucle de jeu, vous afficherez « *Félicitations ! Vous avez trouvé le code !* » ou « *Vous avez perdu...* ».
-  Dans le projet Mastermind, appelez la fonction **runGame()** dans le programme principal.
 - ➡ Activez le projet Mastermind en projet de démarrage puis exécutez-le.
 - ➡ Faites une partie pour vous détendre.


2.10 Un menu principal comme dans les vrais jeux

Nous allons ajouter à notre jeu un menu principal permettant de lancer une nouvelle partie ou d'afficher le tableau des scores (pour le moment inefficace).

-  Ajoutez une fonction **mainMenu()** qui efface le contenu de la console puis affiche le menu suivant et retourne le caractère sélectionné par l'utilisateur.
 - ➡ Elle devra s'assurer que le code saisi est valide.

MASTERMIND	
Menu principal	
1	Nouvelle partie
2	Tableau des scores
q	Quitter
Votre choix :	


 Dans le programme principal de Mastermind, mettez en place une boucle principale qui affiche le menu puis qui exécute les actions adéquates en fonction du code saisi par l'utilisateur.

 A la fin de chaque partie, vous devez demander à l'utilisateur de valider par la touche « entrée » avant de retourner au menu principal.

2.11 Calcul d'un score

Dans notre Mastermind, le score peut être le temps mis par le joueur à déterminer le bon code. En C++, la mesure du temps se fait grâce à la bibliothèque `<chrono>`. Voici un exemple de code permettant de mesurer une durée en secondes :

```
//La ligne suivante stocke dans la variable start un point temporel
auto start = std::chrono::high_resolution_clock::now();
//Ici, c'est le lieu pour lancer un traitement long, comme une partie
//de Mastermind
//La ligne suivante stocke dans la variable finish le point temporel de
//la fin du traitement
auto finish = std::chrono::high_resolution_clock::now();
//La ligne suivante calcule la durée écoulée entre les deux points
//temporels précédents
//La durée est exprimée dans une unité inconnue, propre à
//l'implémentation de la bibliothèque <chrono>
auto duree = finish - start;
//La ligne suivant convertie la durée stockée dans la variable duree en
//secondes.
unsigned long long dureeEnSecondes
= std::chrono::duration_cast<std::chrono::seconds>(duree).count();
```







 Modifiez la fonction `runGame()` pour qu'elle retourne un **unsigned long long** exprimant la durée de la partie en seconde. Dans le cas où le joueur a perdu, la fonction retournera la valeur spéciale `std::numeric_limits<unsigned long long>::max()`.

2.12 Préparation de la gestion des scores

Un score est toujours associé à un nom lorsqu'il est affiché dans le tableau des meilleurs scores. En C++, il est possible d'associer deux types de données ensemble en utilisant le type `std::pair`, qui, comme son nom l'indique permet de créer des paires de données. Ce type s'utilise comme dans l'exemple ci-dessous :



```
//La ligne suivante permet de créer une variable laPaire qui est
//constituée de deux éléments : une chaîne de caractères et un entier non
//signé
std::pair<std::string, unsigned long long> laPaire;
//La ligne suivante permet d'accéder au premier élément, c'est-à-dire la
//chaîne de caractères
laPaire.first = "Super Man";
//La ligne suivante permet d'accéder au second élément, c'est-à-dire
//l'entier non signé
laPaire.second = 42;
```

Comme nous allons devoir gérer un tableau des meilleurs scores, il est souhaitable de créer un alias du type `std::pair<std::string, unsigned long long>` que l'on nommera `score_t`.

-  **Ajoutez au projet Mastermind** un nouveau binôme de fichier `scores.h` / `scores.cpp`.
 Ces fichiers contiendront toutes les fonctions de gestion logique des scores.
-  Créez-y l'alias de type `score_t`.
-  Créez-y également un alias `scoreTable_t` vers le type tableau de `score_t`. Ce tableau aura toujours au maximum 10 éléments. Il peut donc être choisi d'utiliser un tableau de taille fixe.
-  **Ajoutez au projet Mastermind** un nouveau binôme de fichier `scores_cui.h` / `scores_cui.cpp`.
 Ces fichiers contiendront toutes les fonctions d'affichage ou de saisie à la console des scores.

2.13 Vérification de la nécessité d'enregistrer un score

Si le score du joueur est inférieur au score du joueur le moins bon dans le tableau de score, alors il doit être stocké dans ce tableau. Pour rappel, le score est une durée de jeu, le meilleur joueur étant celui qui obtient le temps le plus court.

-  Créez une fonction `higherScore()` qui prend en paramètre la valeur du score du joueur et le tableau des scores. Elle retournera un booléen égal à `true` si ce score mérite d'être dans le tableau, ou `false` sinon.
-  **Dans le programme de Tests** (que vous activerez comme projet de démarrage), tester cette fonction avec les valeurs suivantes :

Nom	Score
Benjamin ALBOUY-KISSI	1
Super Man	2
Wonder Woman	3
Bat Girl	4
Elephant Man	5
Batiman	6
Bat Man	7
Spider Man	8
Je s'appelle Groot	9
Jean-Christophe	10

Valeur du score	Résultat attendu
17	false
10	false
7	true

Assurez-vous également que si le tableau de scores contient moins de 10 scores, le résultat soit toujours `true`.

Vous devriez avoir une sortie du type de celle de la figure suivante :

```
Test de la fonction higherScore() :
un score de 17 : 0
un score de 10 : 0
un score de 7 : 1
un score de 18446744073709551615 dans un tableau non complet : 1
```

2.14 Saisie du nom du joueur

Lorsque le score du joueur doit être inséré dans le tableau des meilleurs scores, il faut qu'il saisisse son nom. Sachant que le tableau des scores sera enregistré dans un fichier CSV par la suite, le nom du joueur ne doit pas contenir de virgule.

Dans le binôme de fichiers adéquat, créez une fonction **askForName()** qui demandera la saisie du nom du joueur et qui retournera ce nom en ayant pris soin de retirer les éventuelles virgules (remplacées par des espaces).

Testez cette fonction dans le projet *Tests*.

Vous devriez avoir une sortie du type de celle de la figure suivante :

```
Test de la fonction askForName() :
Veuillez entrer votre nom : Je,s'appelle,Groot
Vous avez saisi : Je s'appelle Groot
```

Dans cette figure, la sortie console de la fonction **askForName()** est sur fond bleu, les caractères orange sont saisis par l'utilisateur.

2.15 Affichage du tableau de score

Créez une méthode **showScores()** prenant en paramètre une table de scores (de type **scoreTable_t**) et qui l'affichera dans la console comme sur les captures ci-dessous.


MASTERMIND	
Tableau des scores	
Nom	Temps
Benjamin ALBOU	1s
Super Man	2s
Wonder Woman	3s
Bat Girl	4s
Elephant Man	5s
Batiman	6s
Bat Man	7s
Spider Man	8s
Je s'appelle G	9s
Jean-Christoph	10s


MASTERMIND	
Tableau des scores	
Nom	Temps
Benjamin ALBOU	1s
Super Man	2s
Wonder Woman	3s
Bat Girl	4s
Elephant Man	5s
Batiman	6s
-----	----
-----	----
-----	----
-----	----

Testez cette fonction dans le programme *Tests*.

Notez que les noms doivent être tronqués à 14 caractères.

2.16 Ajout d'un score dans le tableau des scores

 Ajoutez une fonction **appendScore()** qui prend en paramètre un score à ajouter (une paire nom \leftrightarrow valeur du score) et un tableau de score. Elle ajoutera le score au tableau à la bonne place (trié par valeur de score) et retournera le nouveau tableau.

 Testez votre fonction dans le programme *Tests* avec les tableaux suivants :

Nom	Score
Benjamin ALBOUY-KISSI	1
Super Man	2
Wonder Woman	3
Bat Girl	4
Elephant Man	5
Batiman	6
Bat Man	7
Spider Man	8
Je s'appelle Groot	9
Jean-Christophe	10



Nom	Score
Starlord	5



Nom	Score
Benjamin ALBOUY-KISSI	1
Super Man	2
Wonder Woman	3
Bat Girl	4
Elephant Man	5
Starlord	5
Batiman	6
Bat Man	7
Spider Man	8
Je s'appelle Groot	9

Nom	Score
Benjamin ALBOUY-KISSI	1
Super Man	2
Wonder Woman	3



Nom	Score
Starlord	5



Nom	Score
Benjamin ALBOUY-KISSI	1
Super Man	2
Wonder Woman	3
Starlord	5

2.17 Intégration des scores dans le jeu Mastermind

- Dans le programme principal du jeu Mastermind, ajoutez une variable pour stocker le tableau des scores (donc de type **scoreTable_t**)
 - ➡ À la fin de chaque partie (après l'appel de la fonction **runGame()**) ajoutez un test pour savoir si la partie est gagnée ou non. Pour rappel, la fonction **runGame()** retourne le score du joueur ou **std::numeric_limits<unsigned long long>::max()** dans le cas où le joueur a perdu.
 - ➡ Dans le cas où la partie est gagnée, vérifiez si le score doit être ajouté au tableau de score en utilisant la fonction **higherScore()**.
 - ➡ Dans le cas où le score doit être ajouté, demandez le nom du joueur à l'aide de la fonction **askForName()** puis ajoutez le nouveau score au tableau à l'aide de la fonction **appendScore()**.
 - ➡ En réponse au cas '2' du menu principal, afficher le tableau des scores en appelant la fonction **showScores()**.
 - ➡ Avant de retourner à l'affichage du menu principal, demandez à l'utilisateur de valider par « entrée ».
- Testez votre jeu en activant le projet Mastermind comme projet de démarrage.
 - ➡ Détendez-vous en jouant une partie.

2.18 Enregistrement des scores dans un fichier

Afin de rendre les scores persistants (c'est-à-dire qu'ils soient mémorisés même quand le jeu est quitté), nous allons les enregistrer dans un fichier CSV qui sera rechargé à chaque lancement du jeu.

- Créez une fonction **writeScores()** qui prendra en paramètre un tableau de scores (de type **scoreTable_t**).
 - ➡ Cette fonction enregistrera dans le fichier *scores.csv* le tableau de score au format CSV.
- Créez une fonction **readScores()** qui retournera le tableau de scores lu depuis le fichier *scores.csv*.
- Testez conjointement ces deux fonctions dans le programme *Tests*.
 - ➡ Enregistrez un tableau de scores avec **writeScores()** puis relisez-le à l'aide de la fonction **readScores()**. Vérifiez que le tableau est égal au tableau initial.
- Ajoutez au début du programme Mastermind le chargement des scores et à la fin du programme l'enregistrement des scores.
 - ➡ JOUEZ !