



Centre Informatique pour les **L**ettres  
et les **S**ciences **H**umaines

## Apprendre C++ avec Qt : Leçon 6

### Manipuler du texte

1 - Instancier et initialiser .....	2
2 - Modifier le contenu .....	2
Insérer .....	2
Effacer.....	3
Remplacer .....	3
3 - Concaténer .....	3
4 - Copier le contenu .....	4
Copies forçant la casse .....	4
Copies partielles .....	4
Copies "nettoyées" .....	5
5 - Texte et valeurs numériques .....	5
Valeur numérique d'une chaîne.....	5
Construire la chaîne représentant une valeur numérique .....	5
Construire une chaîne mêlant texte ordinaire et représentation de valeurs numériques ..	6
6 - Obtenir des informations sur le contenu .....	6
Longueur de la chaîne .....	7
Comparer deux chaînes.....	7
Recherche de la position d'une sous-chaîne.....	7
Limiter la recherche.....	7
Ignorer la casse .....	7
Compter le nombre d'occurrences d'une sous-chaîne .....	8
Expressions régulières.....	8
Utilisation .....	8
Ignorer la casse .....	9
Copier le texte trouvé.....	9
7 - Bon, c'est gentil tout ça, mais ça fait déjà 8 pages. Qu'est-ce que je dois vraiment en retenir ? .....	9

Nous avons vu que le type entier nommé `char` est traité d'une façon particulière lorsqu'une valeur doit être représentée à l'écran : au lieu de construire la séquence de caractères qui représente la valeur dans le système positionnel décimal utilisant les chiffres arabes, les fonctions d'affichage utilisent simplement le caractère correspondant à la valeur dans une table arbitraire (en général la table ASCII ou un de ses dérivés).

S'il est effectivement possible de représenter du texte (c'est à dire des séquences de caractères) à l'aide de séries d'objets de type `char`, ce n'est pas très pratique. La manipulation de texte étant un besoin très répandu, de nombreuses équipes de programmeurs se sont attachées à créer des classes adaptées à cet objectif et qui offrent tout à la fois sécurité, efficacité et simplicité aux programmeurs qui les utilisent. La librairie Qt propose, sous le nom de `QString`, une classe de ce genre.

## 1 - Instancier et initialiser

La classe `QString` n'étant qu'une extension du langage fournie par la librairie Qt, la création d'un objet de ce type exige la présence d'une directive

```
#include "QString.h"
```

La définition d'une `QString` peut alors s'accompagner d'une initialisation. La valeur à stocker peut être fournie soit sous la forme d'un **texte littéral**, soit par une **autre QString** :

```
1 const QString codeModule = "INF Z13";
2 QString message = codeModule;
```

En l'absence d'initialisation explicite, la `QString` créée sera vide.

À la différence des types prédéfinis, les classes "bien conçues" pratiquent, dans la mesure du possible, une initialisation implicite qui place les instances dans un état correspondant à ce qu'on peut attendre d'un "nouveau-né" du type concerné. Dans le cas de `QString`, c'est une chaîne de 0 caractère qui a été choisie comme valeur initiale par défaut.

## 2 - Modifier le contenu

Une fois définies, les `QString` non constantes se prêtent à l'**affectation** de valeurs à partir d'objets de l'un des types utilisables pour l'initialisation ou de valeurs de type `char` :

```
3 message = "Vous avez réussi !";      //texte littéral
4 message = codeModule;                //autre QString
5 message = 'W';                       //char
```

Outre l'opérateur d'affectation, un certain nombre de fonctions membre de la classe `QString` ont pour effet de modifier le contenu de l'instance au titre de laquelle elles sont invoquées.

### Insérer

La fonction `fill()` permet de donner à une `QString` un contenu composé d'un unique **caractère** (indiqué par son premier argument), répété un certain **nombre de fois** (indiqué par son second argument).

```
1 QString lignePointillee;
2 lignePointillee.fill('-', 23);
//lignePointillee contient donc "-----"
//positions dans la chaîne : 0...'...1...'...2...'...3
```

La fonction `insert()` permet d'injecter du texte dans une `QString`, à une **position** déterminée par son premier argument, le second indiquant quel est le **texte à injecter** :

```
1 QString phrase = "Il était une fois une jolie princesse";
// 0...'...1...'...2...'...3...'...4
2 phrase.insert(20, "très");
//phrase contient donc "Il était une fois une très jolie princesse"
// 0...'...1...'...2...'...3...'...4.
```

En C++, toutes les numérotations commencent à zéro (et non à un).

## Effacer

La fonction `remove()` permet d'éliminer certains caractères d'une `QString`. Son premier paramètre indique la **position** du premier caractère qui doit disparaître, alors que le second indique le **nombre** de caractères qui seront supprimés :

```
1 QString phrase = "Il était une fois une très jolie princesse";
  //           0....'....1....'....2....'....3....'....4.
2 phrase.remove(21, 5);
  //On revient donc à "Il était une fois une jolie princesse"
  //           0....'....1....'....2....'....3....'....4
```

La fonction `truncate()` élimine de la `QString` au titre de laquelle elle est appelée tout caractère figurant après la **position** indiquée par son argument.

```
3 phrase.truncate(16);
4 //phrase contient maintenant "Il était une fois"
```

## Remplacer

La fonction `replace()` combine les effets des fonctions `remove()` et `insert()` : ses deux premiers arguments indiquent la **position** et la **longueur** du segment supprimé, alors que le troisième indique le **texte à injecter** à cette même position :

```
1 QString phrase = "Il était une fois une jolie princesse";
  //           0....'....1....'....2....'....3....'....4
2 phrase.replace(21, 5, "affreuse");
  //phrase contient maintenant "Il était une fois une affreuse princesse"
  //           0....'....1....'....2....'....3....'....4
```

La fonction `fill()`, utilisée avec un seul argument, permet de remplacer chacun des caractères présents d'une `QString` par un même caractère (indiqué par son argument).

```
3 phrase.fill('x');
  //phrase contient maintenant "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
  //           0....'....1....'....2....'....3....'....4
```

## 3 - Concaténer

Le verbe "concaténer" est utilisé en informatique pour désigner l'action consistant à joindre bout à bout deux fragments de texte pour créer un texte unique<sup>1</sup>. La classe `QString` respecte la tradition qui veut que cette opération soit symbolisée de la même façon que l'addition :

```
1 const QString codeModule = "INF Z13";
2 QString message = "Vous avez réussi " + codeModule;
3 //message contient "Vous avez réussi INF Z13"
```

En dépit de sa simplicité, l'opérateur de concaténation appelle deux remarques :

La concaténation n'est possible que si l'un des arguments est de type `QString`.

Les expressions suivantes ne permettent donc pas d'obtenir l'effet escompté :

```
"Il était " + "une fois"; //deux chaînes littérales ne se laissent pas concaténer
"la lettre " + 'x';       //une chaîne littérale et un char non plus
```

A la différence de l'addition, la concaténation n'est pas commutative.

La séquence d'instructions suivante donne donc des contenus différents à `tUn` et `tDeux` :

```
4 QString depart = "texte";
5 QString tUn    = depart + 'X'; //un contient maintenant "texteX"
6 QString tDeux  = 'X' + depart; //deux contient maintenant "Xtexte"
```

<sup>1</sup> On pourrait utiliser le verbe "abouter", mais "to concatenate" existe en anglais, et l'emploi de "concaténer" augmente donc le parallélisme entre les deux langues.

## 4 - Copier le contenu

Lorsqu'un programme dispose d'un texte, il peut avoir besoin d'en recopier des portions définies de diverses façons. La classe `QString` propose un certain nombre de fonctions membre qui permettent ce genre d'opérations.

Toutes ces fonctions sont sans effet et renvoient la copie demandée.

En d'autres termes, les `QString` au titre desquelles elles sont appelées ne sont pas modifiées par leur exécution. Pour obtenir un effet sur l'instance utilisée pour appeler la fonction, il faut donc lui affecter la valeur renvoyée par cette dernière.

### Copies forçant la casse

La fonction `upper()` produit une `QString` qui est une version CAPITALISÉE de la `QString` au titre de laquelle elle est appelée :

```
1 QString phrase = "Il était une fois une jolie princesse";
  //           0....'....1....'....2....'....3....'....4
2 QString capitales = phrase.upper();
  //capitales contient "IL ÉTAIT UNE FOIS UNE JOLIE PRINCESSE"
```

La fonction `lower()` produit une `QString` qui est une version dé-capitalisée de la `QString` au titre de laquelle elle est appelée :

```
3 QString minuscules = capitales.lower();
  //minuscules contient "il était une fois une jolie princesse"
```

Remarquez que les fonctions `upper()` et `lower()` conservent les accents, ce qui ne produit pas forcément la capitalisation souhaitée par les francophones.

### Copies partielles

La fonction `left()` renvoie une `QString` dont le contenu est identique au début de celle au titre de laquelle elle est invoquée. Cette fonction exige un argument, qui lui indique la **longueur** de la chaîne qu'elle doit renvoyer :

```
4 QString debut = phrase.left(5); //debut contient "Il ét"
```

La fonction `right()` renvoie une `QString` dont le contenu est identique à la fin de celle au titre de laquelle elle est invoquée. Cette fonction exige un argument, qui lui indique la **longueur** de la chaîne qu'elle doit renvoyer :

```
5 QString fin = phrase.right(6); //fin contient "ncesse"
```

La fonction `mid()` renvoie une `QString` dont le contenu est une copie partielle de celle au titre de laquelle elle est invoquée. Cette fonction exige deux arguments, qui lui indiquent à quelle **position** doit commencer la copie et quelle est la **longueur** de la chaîne qu'elle doit renvoyer :

```
6 QString extrait = phrase.mid(3, 5); //extrait contient "était"
```

Lorsque la copie souhaitée ne concerne qu'un unique caractère, il n'est pas nécessaire d'utiliser la fonction `mid()` : il suffit d'indiquer entre **crochets** la position du caractère visé. Il faut cependant savoir que l'objet ainsi désigné n'est pas de type `char`, mais de type `QChar`.

```
7 QString phrase = "Il était une fois une jolie princesse";
  //           0....'....1....'....2....'....3....'....4
8 QChar uneLettre = phrase[10]; //uneLettre contient 'n'
```

Le type `char` n'offre que 256 valeurs différentes, ce qui pose des problèmes insurmontables pour représenter, par exemple, un texte comportant des passages en russe (alphabet cyrillique), d'autres en arabe, et d'autres encore en chinois, en sanscrit et en hiéroglyphes égyptiens, le tout étant accompagné de transcriptions phonétiques. Les `QString` n'utilisent donc pas les `char` et le code ASCII, mais des `QChar` et Unicode. En échange de cette puissance, une opération de **conversion explicite** est exigée pour retrouver un `char` à partir d'une `QString` contenant du texte "ordinaire" (ie. utilisant l'alphabet latin) :

```
char unChar = phrase[3].latin1();
```

## Copies "nettoyées"

La fonction `stripWhiteSpace()` élimine tous les caractères de type **tabulation**, **espace** ou **saut de ligne** qui figurent en début ou en fin de la `QString` au titre de laquelle elle est appelée :

```
1 QString texte = "\\tOsez, osez, Joséphine. \\n";
2 texte = texte.stripWhiteSpace();
//texte contient maintenant "Osez, osez, Joséphine."
```

La fonction `simplifyWhiteSpace()` effectue le même traitement que `stripWhiteSpace()`, mais le complète en remplaçant par un unique espace toute séquence de caractères de type **tabulation**, **espace** ou **saut de ligne** consécutifs.

```
1 QString suite = "Plus\\trien n's'oppose\\nà la nuit";
2 suite = suite.simplifyWhiteSpace();
//suite contient maintenant "Plus rien n's'oppose à la nuit"
```

## 5 - Texte et valeurs numériques

Les quantités numériques manipulées par l'ordinateur n'étant pas représentées par des suites de caractères, il est fréquemment nécessaire de procéder à des changements de représentation pour assurer la communication entre le programme et les êtres humains qui l'utilisent.

Même s'il "savent" que les ordinateurs ont leur propre façon de représenter l'information, les programmeurs débutants ont souvent du mal à tenir effectivement compte de cette connaissance lorsqu'ils écrivent du code. Les erreurs suivantes sont donc fréquentes :

```
//texte et valeurs numériques : le bêtisier
int valeur = 12;
QString texte = 12; //NON : C++ ne génère pas spontanément la chaîne "12" !
texte = "247";
valeur = texte; //NON : C++ ne génère pas spontanément la valeur 247 !
quelqueChose = texte + valeur; //NON : ça ne veut rien dire...
```

Cette dernière erreur est d'autant plus fréquente que le signe `+` désigne à la fois un opérateur applicable aux quantités numériques (addition) et un opérateur applicable aux `QString` (concaténation). Il n'existe cependant aucun opérateur `+` acceptant que l'un de ses opérandes soit une valeur numérique et l'autre une chaîne de caractère ("*concadition*" ?).

## Valeur numérique d'une chaîne

Lorsqu'une `QString` contient une série de chiffres, il est facile de déterminer quelle est la quantité conventionnellement représentée par cette série : les fonctions `toShort()`, `toInt()`, `toLong()`, `toFloat()` et `toDouble()` renvoient cette valeur, représentée dans le type suggéré par leur nom.

```
1 QString chiffres = "123";
2 int n = chiffres.toInt(); //n contient maintenant 123
3 chiffres.replace(1, 1, "."); //chiffres contient maintenant "1.3"
4 double k = chiffres.toDouble(); //k contient maintenant 1.3
5 n = chiffres.toInt(); //n contient maintenant 0
```

Remarquez (ligne 5) que, si la séquence de caractères n'est pas interprétable dans le type que doit renvoyer la fonction, celle-ci renvoie 0.

## Construire la chaîne représentant une valeur numérique

Pour obtenir la représentation d'une quantité numérique sous la forme d'une séquence de caractères (des chiffres), deux méthodes sont disponibles :

- Lorsque la `QString` devant recevoir la représentation du nombre existe déjà, on peut l'utiliser pour appeler la fonction `setNum()`. Le contenu antérieur de la `QString` sera alors remplacé par la représentation de la **valeur** transmise à `setNum()`, dans le système de notation habituel (système positionnel en base 10 utilisant les chiffres arabes) :

```
1 QString aAfficher;
2 aAfficher.setNum(8612.32); //aAfficher contient maintenant "8612.32"
```

- Lorsque aucune variable n'est disponible pour accueillir la chaîne de chiffres représentant le nombre, la classe `QString` permet d'en générer une "à la volée", ce qui permet en particulier d'éviter de créer des variables pour de simples éléments intermédiaires d'un calcul :

```
3  aAfficher = aAfficher + " plus " + QString::number(27.19);
   //aAfficher contient maintenant "8612.32 plus 27.19"
```

Remarquez que la fonction `number()` n'est pas appelée au titre d'une instance de la classe `QString`, mais au titre de cette classe elle-même. Nous étudierons ce type de fonctions dans une prochaine Leçon. Si la classe `QString` n'offrait pas cette possibilité, l'effet de la ligne 3 n'aurait pu être obtenu qu'au prix de :

```
QString temporaire;
temporaire.setNum(27.19);
aAfficher = aAfficher + " plus " + temporaire;
```

Que la `QString` visée existe déjà ou qu'elle doive être créée pour l'occasion, un second argument permet, si la **valeur** à représenter est entière, d'utiliser une **base** autre que 10 :

```
4  aAfficher.setNum(1789, 16); //aAfficher contient maintenant "6fd"
5  aAfficher = aAfficher + " et " + QString::number(1789, 7);
   //aAfficher contient maintenant "6fd et 5134"
```

### Construire une chaîne mêlant texte ordinaire et représentation de valeurs numériques

Il est, bien entendu, possible de créer ce genre de chaînes "composites" en générant les chaînes représentant les quantités numériques concernées, puis en concaténant les différents fragments. Cette façon de procéder doit toutefois être évitée, car elle conduit à écrire du code dont la logique dépend étroitement de la syntaxe de la langue utilisée pour communiquer avec les utilisateurs du programme. Traduire un tel programme dans une langue différente de celle pour laquelle il a été initialement conçu exige donc de modifier le code, ce qui ne peut être fait que par un programmeur C++ connaissant la langue en question...

La classe `QString` propose une fonction `arg()` qui produit une copie de l'instance au titre de laquelle elle est invoquée dans laquelle est injectée la représentation de la valeur qu'on lui passe comme **argument**. Les **positions d'insertion** doivent être matérialisées dans la chaîne cible par des marqueurs composés d'un signe % suivi d'un chiffre. Ces chiffres indiquent dans quel ordre les positions seront occupées par les valeurs successivement injectées :

```
1  QString message = "Vous avez gagné %1 billes et %2 bons points";
2  int nb = 7;
3  message = message.arg(nb);
   //message contient maintenant " Vous avez gagné 7 billes et %2 bons points"
4  message = message.arg(5);
   //message contient maintenant " Vous avez gagné 17 billes et 5 bons points"
```

L'intérêt de cette façon de procéder apparaît lorsqu'on imagine une modification du message qui implique une remise en cause de l'ordre des éléments. Il suffit, par exemple, de modifier ainsi le contenu initial de la `QString` :

```
5  QString message = "%2 bons points et %1 billes vous avez gagnés";
```

pour que le code précédent produise un message semblant venir de maître Yoda : "5 bons points et 17 billes vous avez gagnés".

Dans un programme réel, les messages ne seront pas spécifiés littéralement dans le code source, mais regroupés dans un fichier texte lu par le programme. L'utilisation de la fonction `arg()` permet donc de changer la langue utilisée par le programme en traduisant ce fichier texte, sans avoir à modifier le code source et à recompiler. Mieux encore, le programme peut laisser le choix de la langue en permettant simplement de choisir le fichier utilisé.

## 6 - Obtenir des informations sur le contenu

Un programme est souvent conduit à s'interroger sur le contenu d'une `QString`. Cette classe propose un grand nombre de fonctions membre permettant d'obtenir diverses informations sur le contenu courant d'une instance.

### Longueur de la chaîne

La fonction `isEmpty()` renvoie un booléen indiquant si la chaîne au titre de laquelle elle est appelée est vide :

```
6 if (message.isEmpty())
7     message = "message était vide";
```

La fonction `length()` renvoie un entier non-signé qui correspond au nombre de caractères contenus dans la chaîne au titre de laquelle elle est appelée :

```
8 unsigned int nbLettres = message.length();
```

La fonction `isEmpty()` n'est donc qu'une façon de vérifier directement si la chaîne est de longueur nulle.

### Comparer deux chaînes

Deux `QString`, ou un `QString` et un texte littéral, peuvent être comparés à l'aide des opérateurs habituels. Les comparaisons sont basées sur l'ordre alphabétique, les majuscules précédant 'a' et les minuscules accentuées venant après 'z'.

```
9 if (message >= "A" && message <= "Z")
10     message = "message commence par une majuscule non accentuée";
```

### Recherche de la position d'une sous-chaîne

La fonction `find()` renvoie la position, dans la chaîne au titre de laquelle elle est appelée, de la première occurrence du texte qui lui est transmis comme argument :

```
1 QString phrase = "Il était une fois une jolie princesse";
//           0....'....1....'....2....'....3....'....4
2 int position = phrase.find("une"); //position contient 9
```

La fonction `findRev()` renvoie la position, dans la chaîne au titre de laquelle elle est appelée, de la dernière occurrence du texte qui lui est transmis comme argument :

```
3 int positionR = phrase.findRev("une"); //positionR contient 18
```

Lorsque la sous-chaîne recherchée n'existe pas `find()` et `findRev()` renvoient -1

### Limiter la recherche

La fonction `find()` peut recevoir un deuxième argument, qui lui indique à quelle position doit commencer la recherche :

```
4 int position2 = phrase.find("une", 10); //position2 contient 18
```

Si la recherche commence en position 10, le premier "une" rencontré est celui de "une jolie".

Dans le cas de la fonction `findRev()`, le deuxième argument indique où la recherche doit s'arrêter. La valeur qui doit être transmise n'est pas une position absolue dans la chaîne, mais une position relative, par rapport à la fin de la chaîne :

```
5 int positionR2 = phrase.findRev("une", -20); //positionR2 contient 9
```

Si la recherche s'arrête 20 caractères avant la fin de la chaîne, le dernier "une" trouvé est celui de "une fois".

### Ignorer la casse

Les fonctions `find()` et `findRev()` peuvent aussi recevoir `false` comme troisième argument, pour leur indiquer qu'elles doivent considérer les versions majuscule et minuscule d'un même caractère comme étant identiques :

```
6 int position3 = phrase.find("UNE", 0, false); //position3 contient 9
```

Du fait des règles d'utilisation des valeurs par défaut des paramètres (Leçon 5), l'utilisation du troisième paramètre exige qu'une valeur soit fournie pour le deuxième.



Notez que, comme la capitalisation des `QString` conserve les accents, ignorer la casse ne rend pas l'accentuation indifférente...

### Compter le nombre d'occurrences d'une sous-chaine

La fonction `contains()` renvoie le nombre d'apparitions, dans la chaîne au titre de laquelle elle est appelée, du texte qui lui est transmis comme argument :

```
7 int nbOccurrences = phrase.contains("i"); //nbOccurrences contient 4
```

Il était une fois une jolie princesse

La fonction `contains()` peut recevoir `false` comme second argument, ce qui lui indique qu'elle doit procéder au dénombrement en ignorant la distinction majuscules/minuscules :

```
8 int nbOccurrences2 = phrase.contains("i", false); //nbOccurrences2 contient 5
```

Il était une fois une jolie princesse

### Expressions régulières

Les fonctions `find()`, `findRev()`, `contains()` et `replace()` acceptent également de recevoir comme premier argument une instance de la classe `QRegExp`. Cette classe fonctionne comme une sorte de classe `QString` pour laquelle certains caractères ou séquences de caractères ont une signification particulière lors des comparaisons avec d'autres chaînes :

	Signification	exemple d'expression	exemples de texte "égaux" à l'expression	exemples de texte "différents" de l'expression
.	un caractère quelconque	a.b	axb	axxb
^	début de chaîne	^ab	abc	xab
\$	fin de chaîne	ab\$	ab	abc
[ ]	tout caractère de l'ensemble décrit entre les crochets	[abc]	a b	d
-	indication d'une plage	[a-z]	b	A
^	inversion de l'ensemble	[^def]	k	e
*	0 ou plus répétitions du caractère précédent	a*b	b aab	a
+	1 ou plus répétitions du caractère précédent	a+b	ab aaab	a b
?	0 ou 1 répétition du caractère précédent	a?b	b ab	aab
\s	séparateur (tabulation, espace, paragraphe...)	a\s b	a b	ab axb

### Utilisation

L'utilisation d'expressions régulières exige la présence d'une directive

```
#include "QRegExp.h"
```

Le passage d'une expression régulière à une fonction membre de la classe `QString` nécessite une instanciation explicite de la classe `QRegExp`, car le simple passage d'un texte littéral donnerait lieu à la transmission d'une valeur de type `QString`. On écrira donc :

```
9 QRegExp expression = "e\\s"; //recherche les mots finissant par 'e'
10 int nbe = phrase.contains(expression); //nbe contient 3
```

Il était une fois une jolie princesse

Remarquez que le symbole `'\'` possède une signification spéciale à deux niveaux : dans la spécification de texte littéral en C++, et en tant que caractère d'une expression régulière. Pour obtenir sa présence dans l'expression, il faut le **redoubler**, faute de quoi la séquence `\s` déclenche un message d'erreur de la part du compilateur, qui essaie sans succès de l'interpréter dans le contexte "texte littéral".



## Ignorer la casse

Si la distinction entre majuscules et minuscules doit être ignorée, c'est à l'expression régulière qu'il faut le dire, et non à la fonction membre de `QString` utilisée pour la recherche :

```
11 QRegExp expression2("e\\s", false); //les mots finissant par 'e' ou 'E'
12 int nbEe = phrase.contains(expression2); //nbEe contient 3
```

Si une instance de `QRegExp` existe déjà, on lui ordonnera de négliger la casse en appelant une fonction membre de cette classe :

```
expression.setCaseSensitive(false);
```

Avec **Qt 2.3**, l'option "ignorer la casse" n'est pas appliquée aux ensembles de caractères, de sorte que :

```
QRegExp expression3 ("[iy]", false);
int nbIY = phrase.contains(expression3); //nbIY contient 4
//Il était une fois une très jolie princesse
```

## Copier le texte trouvé

Lorsqu'on utilise les fonctions `find()` et `findRev()`, on obtient la position à laquelle elles ont découvert un fragment de texte considéré comme "équivalent" à l'expression régulière qu'on leur a transmise. Si l'on souhaite disposer de ce fragment, il manque cependant une information capitale : quelle est sa longueur ?

Par nature, les expressions régulières sont "équivalentes" à des fragments de texte qui peuvent avoir une longueur différente de la leur.

Pour obtenir cette information, il ne faut pas effectuer la recherche avec une fonction membre de `QString`, mais avec une fonction membre de `QRegExp` nommée `match()`. Outre la `QString` sur laquelle doit être effectuée la recherche et la position où la recherche doit commencer, cette fonction peut recevoir comme troisième argument l'adresse d'un `int`, dans lequel elle placera la longueur du fragment "équivalent" découvert :

```
1 QString phrase = "Il était une fois une jolie princesse";
//      0....'....1....'....2....'....3....'....4
2 QRegExp eReg("\\sj[^\\s]*e\\s"); //un mot commençant par j et finissant par e
3 int longueur;
4 int debut = eReg.match(phrase, 0, &longueur);
//debut contient maintenant 21, et longueur 7
5 QString fragment = phrase.mid(debut, longueur);
//fragment contient maintenant " jolie "
```

## 7 - Bon, c'est gentil tout ça, mais ça fait déjà 8 pages. Qu'est-ce que je dois vraiment en retenir ?

- 1) Pour manipuler du texte, on le stocke dans une `QString`.
- 2) Tout fichier `.h` ou `.cpp` qui contient le mot `QString` doit comporter une directive

```
#include "QString.h"
```
- 3) La classe `QString` a de nombreuses fonctions membre et offre beaucoup de possibilités.
- 4) Le programme "[TesteQString](#)" permet de jouer avec plusieurs de ces fonctions, pour mieux comprendre comment on les utilise et quel résultat ou effet elles produisent<sup>2</sup>.
- 5) A part ça, la seule chose à ne pas oublier, c'est l'endroit où on peut retrouver le texte de la Leçon 6.

<sup>2</sup> Si vous recopiez, dans la zone d'édition prévue à cet effet, des expressions régulières citées dans la Leçon 6, veuillez à en éliminer les redoublements de barres obliques exigés par les règles C++ concernant le texte littéral (le contenu d'une zone d'édition n'est pas du code C++...)