

# TP 05 – Chaines de caractères

## Exercice 0 Affichage des accents et autres caractères spéciaux

L'objectif de cet exercice est d'aligner la codification des caractères du code source avec celle utilisée par la ligne de commande. Afin d'avoir un code prêt à affronter l'internationalisation, nous choisirons d'exprimer les caractères selon la norme UTF-8.

Il est possible d'afficher des accents et autres caractères spéciaux dans la console à la condition que le code source et la console soient exprimés dans la même codification des caractères. L'encodage des caractères par défaut dans le code source de Visual Studio est « *Windows 1252* » alors que celui de la console est « *OEM 850* ». Sans rentrer dans les détails, les caractères spéciaux ne sont pas codés avec les mêmes nombres entre ces deux encodages. Il faut alors spécifier l'encodage de nos fichiers sources et aussi celui de la console.

### 0.1 Spécification de l'encodage des fichiers sources

Pour spécifier l'encodage des caractères de vos fichiers sources, dans Visual Studio, choisissez *Fichier / Enregistrer source.cpp sous...* puis dans la boîte de dialogue d'enregistrement, cliquez sur la petite flèche à côté du bouton *Enregistrer*, puis *Enregistrer avec encodage...*

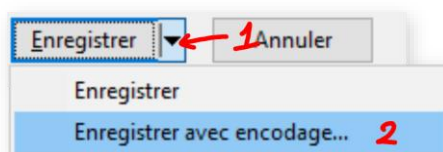


Figure 1 : Enregistrement d'un fichier source avec un encodage spécifique.

Dans la nouvelle boîte de dialogue, choisissez l'encodage *Unicode (UTF-8 sans signature) – Page de codes 65001* et validez par OK.

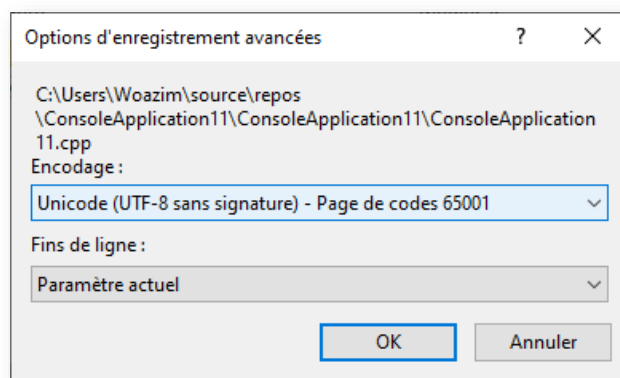


Figure 2 : Sélection de l'encodage.

### 0.2 Spécification de l'encodage de la console de votre programme

La définition de l'encodage de la sortie de la console se fait par l'instruction `SetConsoleOutputCP(CP_UTF8);`. Celle-ci doit être faite avant toute sortie de texte, il est donc logique de l'insérer en tout début de programme. Par ailleurs, pour que cette instruction soit comprise

par le compilateur, il faut inclure la bibliothèque de développement pour Windows en incluant `<Windows.h>`. De même, pour définir l'encodage de l'entrée de la console, il faut utiliser l'instruction `SetConsoleCP(CP_UTF8);`.

Cependant, cette bibliothèque et cette instruction étant spécifiques à Windows, il faut éviter qu'elles ne soient utilisées dans le cas d'une compilation pour un autre système. Pour cela, nous allons utiliser des structures conditionnelles destinées à notre compilateur, plutôt qu'à l'exécution de notre programme. Ces structures sont des sections de code source comprises entre `#ifdef` et `#endif`. Lorsque la compilation se fait pour Windows, les compilateurs définissent une valeur « `_WIN32` » que la structure `#ifdef / #endif` peut identifier. Ainsi, dans le code suivant, les sections entre `#ifdef _WIN32` et `#endif` ne sont compilées que si le système d'exploitation cible est Windows.


```
#ifdef _WIN32
#include <Windows.h>
#endif // _WIN32
#include <iostream>


int main()
{
#ifdef _WIN32
    SetConsoleCP(CP_UTF8);
    SetConsoleOutputCP(CP_UTF8);
#endif // _WIN32
    std::cout << "Des caractères bien représentés 😊\n";
    return 0;
}
```


Ces deux sections de code ne seront compilées que sous Windows

Les « è » et « é » seront correctement affichés dans la console. On peut également afficher des caractères spéciaux comme ce « 😊 »

Code 1 : Spécification de l'encodage de la console pour Windows.

 **ATTENTION** : un bug historique dans la gestion de la ligne de commande de Windows ne permet pas d'entrer des caractères spéciaux via le clavier. Vous ne pouvez donc pas utiliser d'accent lors d'une saisie de texte via `cin`. Cependant, ce bug n'affecte « que » le clavier. Ainsi, la saisie via `cin` reste pleinement fonctionnelle dans le cas de la saisie via *pipe*. Ceci est abordé en cours de système et dans l'exercice 1.4.

 En suivant l'ensemble de ces indications, compilez ce programme pour vous assurer que l'affichage est correct.

 Dorénavant, vous mettrez en œuvre cette technique dans tous vos programmes.

## Exercice 1 Saisie de texte

Dans cet exercice, nous allons faire le tour de toutes les façons de saisir du texte dans vos programmes C++.

### 1.1 Saisie simple

- ✎ Écrivez un programme qui, en utilisant **cin** et un **string**, demande la saisie d'une chaîne de caractères.
  - ➡ Affichez directement la chaîne de caractères pour vérifier votre saisie.
  - ➡ Testez votre programme sans utiliser de caractères spéciaux.
- ✎ À la saisie, entrez une chaîne contenant des espaces
  - ➡ Qu'observez-vous ?

### 1.2 Saisie de chaînes contenant des espaces

Pour résoudre ce problème, après avoir inclus **<iomanip>** vous pouvez utiliser le manipulateur **std::quoted()** comme dans l'exemple ci-dessous.

```
std::string texte;  
std::cout << "Veuillez entrer un texte : ";  
std::cin >> std::quoted(texte);  
std::cout << "Vous avez entré : " << texte << "\n";
```

Code 2 : Exemple de saisie utilisant **std::quoted**.

- ✎ Testez le programme après l'avoir modifié comme dans le code 2. Pour saisir une chaîne de caractères contenant des espaces, vous pouvez encadrer votre saisie par des guillemets doubles « " ». Dans le cas où votre chaîne doit contenir des guillemets doubles, vous devez ajouter « \ » avant chaque guillemet.
  - ➡ Voici quelques exemples de saisies dont vous pourriez observer l'affichage via **cout** :
    - ➡ **Bonjour**
    - ➡ **"Bonjour"**
    - ➡ **Bon"jour"**
    - ➡ **"Bonjour les gens !"**
    - ➡ **"Bonjour les \"gens\" !"**
    - ➡ **"C:\Users\Michel Robert"**
    - ➡ **"C:\\Users\\Michel Robert"**

Le manipulateur **std::quoted()** fonctionne également avec la sortie de texte via **cout**. Dans ce cas, la sortie est automatiquement encadrée de guillemets doubles et les guillemets internes sont échappés avec « \ ».

- ✎ Modifiez le programme pour que la sortie via **cout** soit également manipulée par **std::quoted()** et retestez votre programme.

### 1.3 Saisie d'une ligne complète

Une alternative à la saisie entre guillemets lorsque l'on veut des chaînes contenant des espaces est de lire la ligne complète saisie via **cin**. Pour cela, on peut utiliser **std::getline(std::cin, variable);**.

- ☞ Réalisez la saisie d'un texte via cette méthode puis testez-le avec les mêmes chaines que précédemment.
- ☞ Créez un nouveau programme qui demande l'âge de l'utilisateur, puis son nom, puis affiche ces informations lues.
  - ☞ L'âge sera saisi dans un **unsigned int**.
  - ☞ Le nom sera saisi en utilisant **std::getline()**.
  - ☞ Y a-t-il un problème ? (Si je demande ça, c'est que la réponse est oui)

Lorsqu'on demande la saisie d'un nombre, **cin** va parcourir les caractères saisis un par un et s'arrêter dès qu'un caractère n'est plus compatible avec la lecture d'un nombre. Ce peut être un espace, un saut de ligne ou une lettre. Ainsi, dans le dernier programme, vous avez certainement entré un nombre puis validé par la touche « Entrée ». Cette touche ajoute le caractère de saut de ligne dans le texte de **cin** et stoppe le décodage du nombre. En conséquence, la fonction **std::getline()** commence par recevoir le caractère de saut de ligne en premier lieu et ne décode donc pas plus.

La solution consiste à demander à **cin** d'ignorer les caractères « blancs » avant de décoder une ligne. Cela se fait par le manipulateur **std::ws** qui doit être appliqué à **cin** avant de faire le **std::getline()** comme dans l'exemple suivant.

```
std::cin >> std::ws;  
std::getline(std::cin, texte);
```

ou bien en une seule ligne

```
std::getline(std::cin >> std::ws, texte);
```

Code 3 : Exemple de saisie utilisant **std::getline** après la saisie d'un nombre.

- ☞ Modifiez votre programme avec ces nouvelles connaissances puis testez à nouveau votre programme.

☞ Dorénavant vous utiliserez l'une de ces techniques pour la saisie de texte.

## 1.4 Les caractères spéciaux en entrée

Comme vu en introduction dans la section 0.2, un bug historique de la gestion de la console Windows ne permet pas la saisie de caractères spéciaux via le clavier. Cependant, la connexion des flux du programme via *pipe* est totalement fonctionnelle. Pour rappel, un programme est muni de 3 flux de communication par défaut : **stdin**, **stdout** et **stderr**, correspondant respectivement en C++ à **std::cin**, **std::cout** et **std::cerr**. Sans intervention de votre part, un programme voit son flux **stdin** « connecté » à la saisie au clavier, et ses flux **stdout** et **stderr** « connectés » à l'écran de la console.

À l'aide de commandes spécifiques à l'interpréteur de ligne de commande utilisé (typiquement sous Windows : **cmd.exe** ou **PowerShell**), il est possible de rediriger ces flux vers d'autres commandes ou des fichiers. Les deux exemples ci-dessous permettent de rediriger le contenu d'un fichier *inputUTF8.txt* vers le flux **stdin** du programme **prog.exe**.

```
C:\Prog>prog.exe < inputUTF8.txt
```

Figure 3 : Redirection du contenu du fichier *inputUTF8.txt* vers le flux *stdin* du programme *prog.exe* dans la ligne de commande standard de Windows (*cmd.exe*)

```
PS C:\Prog> Get-Content .\inputUTF8.txt | .\Prog.exe
```

Figure 4 : Redirection du contenu du fichier *inputUTF8.txt* vers le flux *stdin* du programme *prog.exe* dans la ligne de commande PowerShell de Windows

- 📄 Créez un fichier texte en encodage UTF-8 contenant les saisies compatibles avec le programme obtenu à la section 1.3.
  - ➡ Utilisez un éditeur permettant de spécifier l'encodage (par exemple le Bloc-Notes de Windows, Notepad++, ou bien sûr Visual Studio)
  - ➡ Insérez des caractères « spéciaux » pour le nom. Par exemple, votre fichier peut être :

```
93
Édouard
```

Figure 5 : Exemple de contenu de fichier texte au format UTF-8.

- 📄 Via une ligne de commande (*cmd* ou *PowerShell*), exécutez votre programme en redirigeant le contenu du fichier texte créé vers le flux *stdin* de votre programme.
  - ➡ Pour rappel, votre programme est compilé dans le sous répertoire **x64\Debug** de votre solution Visual Studio.
  - ➡ Assurez vous que votre programme affiche bien l'âge et le nom avec les caractères corrects.








- 📄 On est d'accord que ce n'est vraiment pas pratique pour « saisir » des caractères spéciaux. Des techniques existent pour permettre la saisie directement depuis le clavier, mais sont complexes et spécifiques à Windows. Le code que vous avez écrit est totalement compatible avec les autres systèmes d'exploitation.
- 📄 Par la suite, en attendant que Microsoft résolve ce problème, vous ne saisirez jamais de caractères accentués dans vos programmes.

## Exercice 2 Jouons avec les chaines de caractères




### 2.1 Compter le nombre de mots

- 📄 Créez un programme qui demande une chaîne de caractères puis affiche le nombre de mots la composant.
  - ➡ Nous considérerons que les mots sont les sections de la chaîne qui sont séparés par des caractères espaces.
- 📄 Améliorez votre programme pour que si deux mots sont séparés par plusieurs espaces, le compte soit correct.

## 2.2 Chercher du texte

-  Créez un programme qui demande la saisie d'une chaîne de caractères
  -  Vous stockerez cette chaîne dans une variable **texte**.
-  Ajoutez la saisie d'une seconde chaîne de caractères
  -  Vous stockerez cette chaîne dans une variable **recherche**.
-  Écrivez un algorithme capable de trouver la chaîne « **recherche** » dans la chaîne « **texte** »
  -  Votre algorithme affichera l'indice du premier caractère de **recherche** dans **texte** si la chaîne **recherche** a été trouvée, ou bien affichera un message indiquant que la chaîne **recherche** n'a pas été trouvée.
-  Modifiez votre programme pour que si la chaîne **recherche** est trouvée plusieurs fois dans **texte**, alors toutes les positions trouvées soient affichées.

## 2.3 Remplacer du texte

-  Ajoutez au programme précédent la saisie d'une troisième chaîne « **remplace** »
-  Modifiez le programme pour qu'il construise une nouvelle chaîne de caractères **texte2** dont toutes les occurrences de **recherche** auront été remplacées par **remplace**.
  -  Évidemment, vous afficherez la nouvelle chaîne de caractère **texte2**.

## 2.4 Tout ça en plus simple

Le C++, comme beaucoup de langages, vous apporte ces fonctionnalités de recherche et remplacement directement dans le langage. La recherche se fait par l'appel de la fonction **find** comme dans l'extrait de code ci-dessous :

```
size_t pos = texte.find(recherche, 0);
```

Code 4 : Exemple de recherche de la chaîne **recherche** dans la chaîne **texte**.

Dans l'extrait de code 4, la variable **pos** prend la valeur de l'indice de caractère où le premier texte **recherche** est trouvé dans la chaîne **texte**. Si aucune occurrence n'est trouvée, alors la valeur de **pos** est la constante **std::string::npos**. La valeur « 0 » inscrite dans le code 4 correspond à l'indice de caractère à partir duquel commencer la recherche. Elle peut être omise dans le cas où l'indice de départ est 0.

Pour le remplacement de texte, la fonction du C++ qui permet de réaliser cette opération est la fonction **replace**. Elle ne permet pas de chercher et remplacer, elle permet seulement de remplacer une portion de chaîne identifiée par une position et une longueur par une autre chaîne. Elle s'utilise comme dans l'exemple ci-dessous :





```
texte.replace(10, 5, replace);
```

Code 5 : Exemple de remplacement dans la chaîne **texte** par la chaîne **remplace**.

Dans l'extrait de code 5, les caractères 10 à 14 (= 10 + 5 - 1) de la chaîne **texte** seront remplacés par la chaîne **remplace**.





-  À l'aide de ces deux fonctions, reprenez l'exercice 2.3.

### Exercice 3 Les chaines – des données comme les autres

-  Créez un tableau de 12 chaines de caractères initialisé avec les 12 mois de l'année.
-  Demandez le numéro du mois à l'utilisateur
-  Sélectionnez dans le tableau la chaine de caractères correspondante au numéro du mois saisi.
-  Affichez cette chaine.

### Exercice 4 Comparaison de chaines

Les opérateurs de comparaison `<`, `<=`, `>` et `>=` permettent de comparer deux chaines de caractères en les classant dans l'[ordre lexicographique](#). Cependant, cet ordre se fait uniquement de façon numérique sur les codes de caractères et ne correspond donc pas souvent à un ordre alphabétique usuel, par exemple « ZOO » sera inférieur à « alphabet » car le code numérique du « Z » est inférieur au code numérique du « a ».


-  Dans un nouveau programme, créez deux chaines de caractères `mot1` et `mot2` contenant respectivement « ébahi » et « enclave ».
-  A l'aide de l'opérateur `<`, déterminez celle qui est classée avant dans l'ordre lexicographique.
-  Afficher cette comparaison dans la console soit par « "ébahi" < "enclave" » ou « "enclave" < "ébahi" » selon le cas obtenu.
-  Qu'observez-vous ?

Afin d'avoir une comparaison plus « naturelle », il faut faire appel à la capacité d'internationalisation du C++. L'internationalisation, ou « localisation » se fait à l'aide de la bibliothèque `<locale>`. Une comparaison qui respecte la langue humaine se fait par la syntaxe suivante :

```
std::locale("en_US.UTF-8")(mot1, mot2)
```

Code 6 : Comparaison lexicographique de deux chaines `mot1` et `mot2` en respectant la langue anglaise, américaine encodée en UTF-8.

Dans l'extrait de code 6, l'expression écrite retourne **true** si `mot1` est avant `mot2`, **false** sinon. La chaine de caractères « `en_US.UTF-8` » permet de spécifier la localité de la comparaison. Typiquement, pour une comparaison française, on pourrait mettre « `fr_FR.UTF-8` ». Cependant, la comparaison localisée est déléguée au système d'exploitation. Le système doit alors avoir d'installée la localisation demandée. Sous Windows, il n'y a pas trop de problèmes car beaucoup de localisations sont automatiquement préinstallées. Sous Linux, il est très fréquent que seule la localisation de la langue de l'utilisateur soit installée en plus de « `en_US.UTF-8` ». Le choix le plus sûr est donc « `en_US.UTF-8` ».

-  Fort-e de ces nouvelles informations, comparez les deux mots « ébahi » et « enclave » en vous assurant par affichage que « ébahi » est bien avant « enclave ».