# TP 1

## Les string

### Exercice 1 Lecture

*Soit la variable* `name` *de type* `string`. *La commande* `cin >> name` *permet de lire un* `string` *sur l'entrée standard le string et de ranger ce qu'il vient de lire dans la variable* `name`. *Lorsque l'on lit un string, on commence par éliminer les caractères blancs (espaces, tabulation, fin de fichier ...). Puis on lit la chaîne de caractères dans* `name` *jusqu'à rencontrer un nouveau blanc ou une fin de fichier.*

Prédire le comportement du programme suivant lorsque l'on entre deux noms (par exemple : "Rolling Stones"), puis essayer-le.

```
#include <iostream>
#include <string>

int main()
{
   std::cout << "Comment vous appeller vous ? ";
   std::string name;
   std::string firstname;
   std::cin >> name >> firstname;

   std::cout << "Salut " << name << std::endl
                << "Et quel est le votre ? ";
   std::cin >> name;
   std::cout << "Enchante " << name << std::endl;

   return 0;
}
```

### Exercice 2 Ecriture

On veut écrire un programme qui demande le nom de l'utilisateur et affiche le message suivant :

```
******************
*                *
* Bonjour, Enzo ! *
*                *
******************
```

### Question 1

Définir le canevas du message.

### Question 2

Ecrire le programme.

### Question 3

Tester avec les prénom suivant : Léa, Vladimir.

## Exercice 3 Ecriture avec cadre variable

On va vouloir maintenant produire un résultat similaire mais avec un nombre d'espaces blancs entre le cadre et le message.
exemple :

```
 **********************
1*                      *
2*                      *
3*                      *
 *123Bonjour, Enzo !123*
1*                      *
2*                      *
3*                      *
 **********************
```

On peut voir notre sortie comme un tableau de caractères, pour lequel on peut écrire une ligne à la fois.

### Question 1

Soit le nombre de ligne blanche défini de la manière suivante.

```
const int blanc = 2;
```

On souhaite un cadre symétrique, écrire la déclaration et l'initialisation du nombre de lignes.

### Question 2

Ecriture d'une ligne. On peut remarquer que toutes les lignes à écrire aurons la même taille. Si on considère notre affichage comme un tableau de caractères, la taille d'une ligne est le nombre de colonnes. On définit le nombre de colonnes de la manière suivante :

```
const std::string::size_type cols = phrase.size() + blanc*2 + 2;
```

- A quoi sert le const ?
- Pourquoi utiliser le type `std::string::size_type` ?

### Question 3

L'écriture d'une ligne va se faire caractère par caractère.

1. Donner la condition d'écriture du caractère `*`.
2. Donner la condition d'écriture de la phrase `Bonjour, ...`

### Question 4

Ecrire le programme complet.
*Ecrire* `std::` *devant chaque nom est un peu lourd. Pour ne pas avoir à faire cela on peut utiliser des* `using-declaration`*. Cela permet d'expliciter le fait que certains noms utilisés appartiennent toujours au même espace de nom. La porté de cette déclaration est le bloc, on peut par conséquent le définir de manière global avant le main. Exemple :*

```
using std::cout;

using namespace std;
```

*La première ligne défini* `cout` *comme un synonyme de* `std::cout`*. La seconde ligne dit que l'ensemble des noms utilisés appartiennent à l'espace de nom* `std`*.*

## Exercice 4 Lecture/Ecriture dans un fichier

*la bibliothèque* `fstream` *donne la possibilité d'ouvrir (*`open(nom, mode d'ouverture)`*) ou de fermer (*`close()`*) un fichier et de l'utilisé comme un stream. On peut précisier le mode d'ouverture :*

- *lecture :* `fstream::in`
- *écriture :* `fstream::out`

### Question 1

Ecrire un programme qui compte le nombre de mots dans le fichier `travail.txt`. Vérifier le résultat avec la commande UNIX `wc`.

### Question 2

Ecrire un programme qui lit le fichier `prenom.txt` et qui affiche l'ensemble des prénoms par ordre alphabétique dans le fichier `prenom_trier.txt`.

## Exercice 5 Triangle de pascal

Ecrire un programme qui affiche le triangle de pascal pour un nombre de lignes donné par l'utilisateur.

Exemple :

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

Pour une case $(i, j)$ où $i$ est le numéro de ligne et $j$ le numéro de colonne, on a $(i, j) = (i - 1, j - 1) + (i - 1, j)$