Pas à pas 1 : Reprise de l'exemple du cours (lecture et écriture sur fichier, niveau 0)

> Cet exercice est disponible en page 68 de l'ouvrage *C++ par la pratique*

On veut écrire un programme `fichiers.cc` qui permet de lire le contenu d'un fichier et de le copier dans un autre.

Cet exercice reprend pas à pas les différentes étapes pour y parvenir.

1. Comme d'habitude, créez la « coquille » de base de votre programme dans un fichier `fichiers.cc` :

```
#include <iostream>
using namespace std;

int main()
{

  return 0;
}
```

2. Pour utiliser les fichiers, il faut inclure les fichiers de définitions correspondant : `fstream`.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{

  return 0;
}
```

3. Prototypez la fonction `demande_noms` qui va demander deux noms de fichier à l'utilisateur (un pour l'entrée et un pour la sortie).
Essayez de le faire sans regarder la solution ci-dessous. Essayez également de ne rien oublier.

Solution :

On veut demander 2 noms de fichiers... Dans quel type de données allons-nous les mettre ?
Dans des `string`s tout naturellement.
Donc première chose à faire : inclure la bibliothèque des `string`s :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
...
```

Ensuite, quel doit être le prototype demandé ?
La question est : comment cette fonction doit-elle retourner ces 2 informations au reste du programme.

Deux solutions sont envisageables : la plus naturelle serait justement de retourner les deux `string`s via la valeur de retour. Mais une fonction en C++ ne peut retourner qu'un seul élément. On peut donc penser ici à créer une structure qui contiendrait 2 `string`s et retourner cette structure.

Cette solution est tout à fait possible et correcte, mais peut être un peu lourde ici (car cette structure ne servirait plus ensuite).

Une autre solution offerte en C++ est d'utiliser le passage par référence qui permet à une fonction de modifier ses arguments.

Avec cette solution, il faut donc passer 2 `string`s par référence comme arguments à la fonction ; ce qui nous conduit au prototype suivant :

```
void demande_noms(string& fichier_entree, string& fichier_sortie);
```

On a donc à ce stade le code suivant :

```
#include <iostream>
#include <string>
using namespace std;

void demande_noms(string& fichier_entree, string& fichier_sortie);

int main()
{

  return 0;
}
```

---

4. Définissez maintenant cette fonction.
Essayez de le faire sans regarder la solution ci-dessous.

Solution :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void demande_noms(string& fichier_entree, string& fichier_sortie);

int main()
{

  return 0;
}

// Demande les noms des fichiers a l'utilisateur
void demande_noms(string& fichier_entree, string& fichier_sortie) {
  cout << "Saisie des nom de fichiers." << endl;
  cout << "    Entrez le nom du fichier de lecture:  ";
  cin >> fichier_entree;
  cout << "    Entrez le nom du fichier de sortie:  ";
  cin >> fichier_sortie;
}
```

5. Définissez (prototype et définition) une autre fonction `transfert_fichier` qui va copier, ligne par ligne, le contenu du fichier dont le nom est contenu dans `'fichier_entree'` dans le fichier dont le nom est contenu dans `'fichier_sortie'`.
   - Ouvrez les flots d'entrée et de sortie avec les noms de fichier passés en arguments (en C++98, il faut ajouter `c_str()` derrière le nom).
   - Testez l'état du flot avec la fonction `fail()`.
   - Il faut ensuite lire le flot d'entrée ligne par ligne (en utilisant la fonction `getline()`) et le recopier dans le flot de sortie. L'extraction s'arrête lorsque cette lecture est impossible, i.e. lorsque `getline()` retourne `false` (dans un contexte de test).
   - À la fin, il ne reste plus qu'a fermer les flots avec la fonction `close()`.

Essayez de le faire sans regarder la solution ci-dessous.

Solution :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

...
void transfert_fichier(string fichier_entree, string fichier_sortie);

int main()
{
```

```
  return 0;
}

...

 // transfère les données du fichier 'fichier_entree' au fichier 'fichier_sortie'
void transfert_fichier(string fichier_entree, string fichier_sortie) {

  // ouvre le flot d'entrée
  ifstream entree(fichier_entree);
  // en C++98 (i.e. avant C++11) : ajouter c_str() :
  //   ifstream entree(fichier_entree.c_str());

  // vérifie que le fichier existe
  if (entree.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_entree
         << " en lecture." << endl;
    return;
  }

  // ouvre le flot de sortie
  ofstream sortie(fichier_sortie); // en C++98 : ajouter c_str()

  if (sortie.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_sortie
         << "en écriture." << endl;
    return;
  }

  // lit le fichier ligne par ligne jusqu'à la fin de celui-ci
  string ligne;
  while (getline(entree, ligne)) {
     sortie << ligne << endl; // copie la ligne dans le flot de sortie
  }

  // ferme les flots
  entree.close();
  sortie.close();

  cout << "Transfert terminé !"<< endl;
}
```

6. Il ne reste plus qu'à compléter le `main` pour copier les données :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

...

int main()
{
  string fichier_entree, fichier_sortie;

  // demande les noms des fichiers
  demande_noms(fichier_entree, fichier_sortie);

  // transfère les données du fichier 'fichier_entree' au fichier
  // fichier_sortie'
  transfert_fichier(fichier_entree, fichier_sortie);


  return 0;
}

...
```

Programme complet :

(**cliquez ici** pour passer au tutoriel suivant)

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

void demande_noms(string& fichier_entree, string& fichier_sortie);
void transfert_fichier(string fichier_entree, string fichier_sortie);

int main()
{
  string fichier_entree, fichier_sortie;

  // demande les noms des fichiers
  demande_noms(fichier_entree, fichier_sortie);

  // transfère les données du fichier 'fichier_entree' au fichier
  // fichier_sortie'
  transfert_fichier(fichier_entree, fichier_sortie);

  return 0;
}

// demande les noms des fichiers à l'utilisateur
void demande_noms(string& fichier_entree, string& fichier_sortie){
  cout << "Saisie des nom de fichiers" << endl;
  cout << "    Entrez le nom du fichier de lecture:  ";
  cin >> fichier_entree;
  cout << "    Entrez le nom du fichier de sortie:  ";
  cin >> fichier_sortie;
}

// transfère les données du fichier 'fichier_entree' au fichier 'fichier_sortie'
void transfert_fichier(string fichier_entree, string fichier_sortie) {

  // ouvre le flot d'entrée
  ifstream entree(fichier_entree);

  // vérifie que le fichier a bien été ouvert
  if (entree.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_entree
         << " en lecture." << endl;
    return;
  }

  // ouvre le flot de sortie
  ofstream sortie(fichier_sortie); // en C++98 : ajouter c_str()
  if (sortie.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_sortie
         << " en écriture " << endl;
    return;
  }

  // lit le fichier ligne par ligne jusqu'à la fin de celui-ci
  string ligne;
  while (getline(entree, ligne)) {
     sortie << ligne << endl; // copie la ligne dans le flot de sortie
  }

  // ferme les flots
  entree.close();
  sortie.close();

  cout << "Transfert terminé !"<< endl;
}
```

Pas à pas 2 : Reprise de l'exemple du cours (Manipulateurs, niveau 0)

> Cet exercice est disponible en page 72 de l'ouvrage *C++ par la pratique*

On veut écrire un programme `manipulateurs.cc` qui permet d'écrire une matrice identité (avec que des 1 sur la diagonale et des 0 ailleurs) dans un fichier, en utilisant les manipulateurs de flot.

Cet exercice reprend pas à pas les différentes étapes pour y parvenir.

1. Ouvrez le fichier (vide) `manipulateurs.cc` dans votre éditeur favori.
2. Préparez la « coquille » vide de base accueillant votre programme. Pour utiliser les fichiers, il faut inclure le fichier de définitions correspondant , `fstream`.

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{

  return 0;
}
```

3. Définissez (prototype et définition) une fonction `ecrit_matrice_identite` qui aura pour but d'écrire la matrice désirée (indiquée par sa taille, passée en argument) dans un fichier (dont le nom est passé en argument).
   - Ouvrez le flot de sortie et testez l'ouverture avec la fonction `fail()` ;
   - Écrivez la matrice dans le fichier ouvert, en utilisant les manipulateurs `setw()` et `setfill()`. Ces fonctions nécessitent d'inclure le fichier de définitions `iomanip`.
     Note: `setfill()` n'a pas été présentée formellement en cours; elle permet simplement de changer le caractère utilisé pour remplir le « vide » nécessaire; p.ex. `cout << setfill('_');` fera que les prochains remplissages seront effectués avec des _ ;
   - À la fin, il ne reste plus qu'à fermer le flot avec la fonction `close()`.

   Essayez de le faire sans regarder la solution ci-dessous.

   Solution :

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

...
void ecrit_matrice_identite(string fichier_sortie, size_t taille);

int main()
{

  return 0;
}

// écrit la matrice identité dans un fichier
void ecrit_matrice_identite(string fichier_sortie, size_t taille){

  // ouvre le flot
  ofstream sortie(fichier_sortie); // en C++98 : ajouter c_str()

  if (sortie.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_sortie
  << "en écriture." << endl;
    return;
  }

  sortie << setfill('0') ;
```

```
  for (size_t i(1); i < taille; ++i)
    sortie << setw(i) << '1' << setw(taille-i) << '0' << endl;
  sortie << setw(taille) << '1' << endl;

  // ferme le flot
  sortie.close();
  cout << "Matrice sauvegardée dans le fichier " << fichier_sortie << "."
       <<  endl;
}
```

Notez bien que les manipulateurs `setw` n'affichent rien en eux-même mais influencent le *prochain* affichage ! Il est donc nécessaire d'afficher quelque chose après le `setw`, par exemple ici le dernier `'0'` (ou le dernier `'1'` pour la dernière ligne).

4. Il ne reste plus qu'à compléter le `main` pour créer la matrice, par exemple comme ici (d'autres variantes sont possibles) :

```
...

int main()
{
  size_t taille(0);
  string fichier_sortie;

  cout << "Taille de la matrice :" << endl;
  cin >> taille;
  if (taille > 25) taille = 25;
  else if (taille < 2) taille = 2;

  cout << "Nom du fichier de sortie :";
  cin >> fichier_sortie;

  // écrit la matrice identité dans le fichier
  ecrit_matrice_identite(fichier_sortie, taille);

  return 0;
}

...
```

Programme complet :

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;

void ecrit_matrice_identite(string fichier_sortie, size_t taille);

int main()
{
  size_t taille(0);
  string fichier_sortie;

  cout << "Taille de la matrice :" << endl;
  cin >> taille;
  if (taille > 25) taille = 25;
  else if (taille < 2) taille = 2;

  cout << "Nom du fichier de sortie :";
  cin >> fichier_sortie;

  // écrit la matrice identité dans le fichier
  ecrit_matrice_identite(fichier_sortie, taille);
```

```
  return 0;
}

// écrit la matrice identité dans un fichier
void ecrit_matrice_identite(string fichier_sortie, size_t taille){

  // ouvre le flot
  ofstream sortie(fichier_sortie); // en C++98 : ajouter c_str()

  if (sortie.fail()){
    cerr << "Erreur : impossible d'ouvrir le fichier " << fichier_sortie
  << "en écriture." << endl;
    return;
  }

  sortie << setfill('0') ;
  for (size_t i(1); i < taille; ++i)
    sortie << setw(i) << '1' << setw(taille-i) << '0' << endl;
  sortie << setw(taille) << '1' << endl;

  // ferme le flot
  sortie.close();
  cout << "Matrice sauvegardée dans le fichier " << fichier_sortie << "."
       <<  endl;
}
```

Dernière mise à jour le 3 décembre 2015
Last modified: Thu Dec 3, 2015