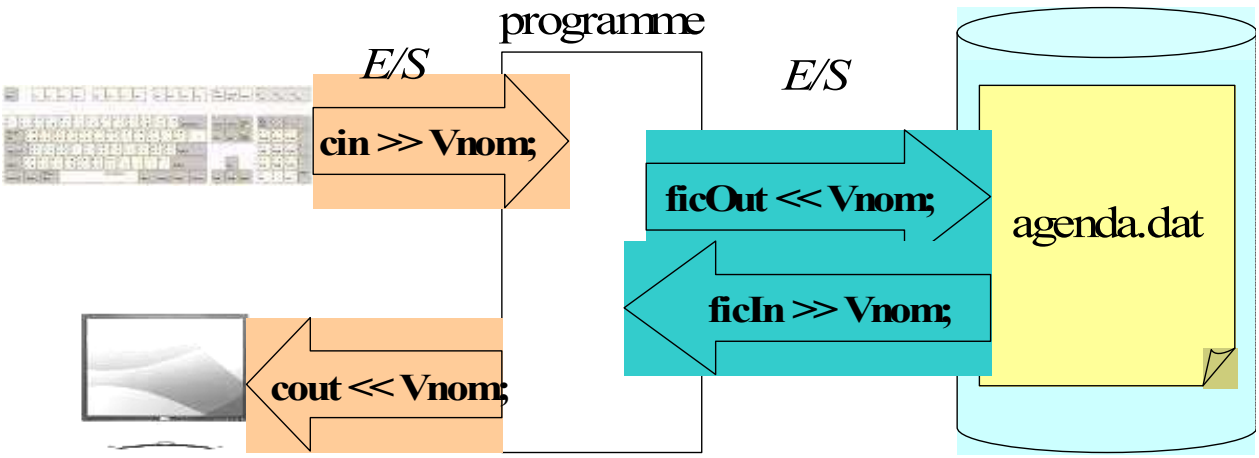
C++

Ch 6 – Fichiers

I. Introduction

A. Flux d'Entrées-Sorties

Les flux (*anglais : stream*) correspondent aux échanges réalisés entre un programme et les périphériques : clavier, écran, disque dur, etc.



1. Flux d'Entrées-sorties clavier/écran

Les entrées-sorties sont toujours réalisées comme des flots de données (flux) qu'on peut symboliser comme des canaux entre le programme en mémoire et les différents périphériques :

- Clavier → programme : flux d'entrée **cin**, associé à l'opérateur >>
  - Lecture à partir du clavier et « envoi » vers une variable
- Programme → écran : flux de sortie **cout** associé à l'opérateur <<
  - Ecriture du contenu d'une variable vers l'écran

2. Flux d'Entrées-sorties disque : les fichiers

Les fichiers permettent la conservation des informations saisies de manière permanente (on parle de persistance des informations) sur un support magnétique afin de pouvoir les réutiliser plus tard.
Comme la lecture du clavier et l'écriture vers l'écran, les flux disques vont également dans les 2 sens :

- Disque dur → programme : flux d'entrée associé à l'opérateur >>
- Programme → disque dur : flux de sortie associé à l'opérateur <<

L'utilisation d'un fichier comporte en général 3 phases :

- L'ouverture d'un flux selon un certain mode (lecture, écriture, etc)
- La lecture ou l'écriture selon le mode
- La fermeture du flux.

B. Librairie C++ pour les Flux d'Entrées-Sorties fichiers

Pour utiliser les fichiers en C++, il est nécessaire d'inclure le fichier d'en-tête **<fstream >.**

II. Fichiers textes

Les fichiers textes sont constitués uniquement de caractères ASCII visibles (mis à part les caractères de fin de ligne : CR LF (Carriage Return, Line Feed), OD OA en hexadécimal.
La longueur de chaque ligne dépend de la longueur de chacune des valeurs qui a été enregistrées. Chaque valeur est séparée par un espace qui va permettre la relecture (comme les espaces

Un fichier texte peut être ouvert par un éditeur de texte : mais attention aux modifications qui pourraient nuire à la relecture du fichier par le programme.

A. Flux et fonction de gestion des fichiers textes

flux		***fonction***	
ifstream	Flux d'entrée		
ofstream	Flux de sortie		
		f.open(x)	Pour ouvrir un flux nommé x (nom du fichier)
		f.close()	Pour fermer un flux
		f.eof()	Pour texter la fin d'un flux d'entrée
		f.fail()	Pour tester la bonne exécution de la dernière action sur un flux
		getline(f,s)	Pour une ligne compléte d' un flux f vers une chaîne (string)

B. Flux de sortie : ofstream

ATTENTION :

L'ouverture d'un flux de sortie créé le fichier s'il n'existe pas MAIS L'ECRASE S'IL EXISTE DEJA.

Syntaxe:

```
ofstream outFic1(nom_de_fichier);
```

ou bien :

```
ofstream outFic1 ;
outFic1.open(("fichier.txt") ;
```

- **outFic1:** nom logique attribué au flux
- **nom_de_fichier :** nom du fichier physique (chemin d'accès, nom, extension)
  - variable, constantes, littéral chaine de caractère
- **inFic1:** nom logique

pour fermer le flux :

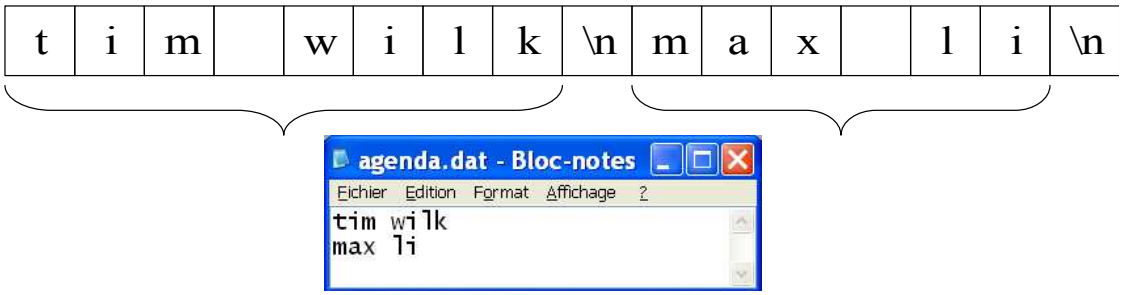
```
outFic1.close() ;
```

- **outFic1:** nom logique

```
ofstream monFichier;
monFichier.open("agenda.dat");

monFichier << "tim" << "  " << "wilk" << endl;
monFichier << "max" << "  " << "li" << endl;

monFichier.close();
```



Exemple :

```
// tableau de caractères équivalent à une chaine de caractères
const char NOM_FIC[]  = "agenda.dat"; // tableau de caractères

ofstream outFic; // déclaration d'un flux de sortie
outFic.open(NOM_FIC);// ouverture du flux

if (outFic.fail())
  cerr << "erreur à l'ouverture"; // envoi vers le flux d'erreur
  else
 {
  outFic << Vprenom <<" "<< Vnom<< endl;
  outFic.close() ;
 }
```

C. Flux d'entrée : ifstream

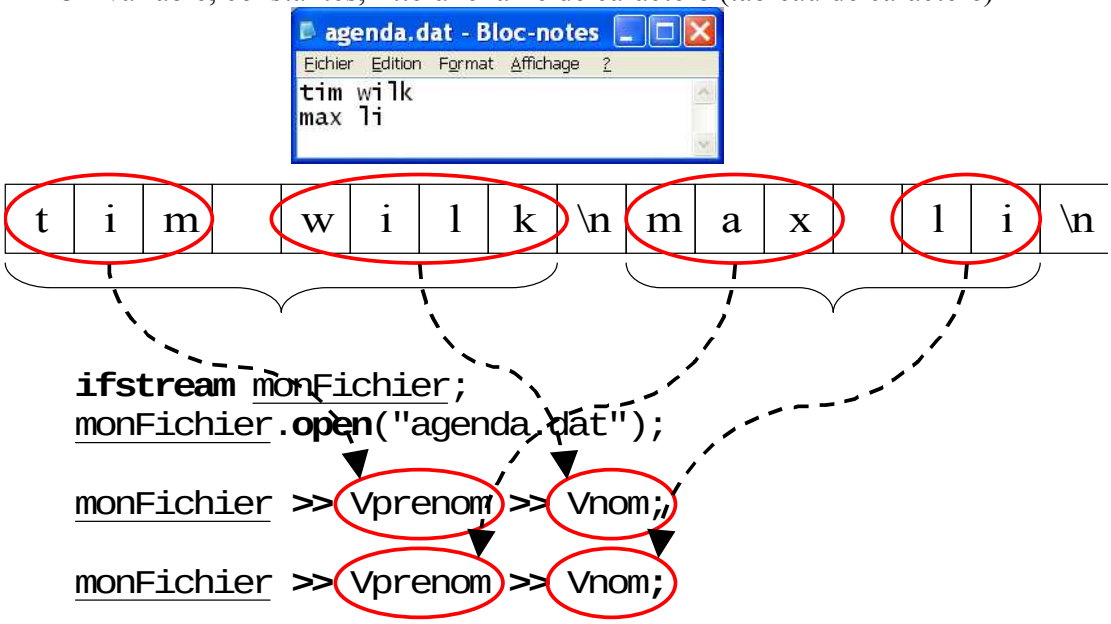
Syntaxe:

```
ifstream inFic1(nom_de_fichier);
```

ou bien :

```
ifstream inFic1 ;
inFic1.open(("fichier.dat") ;
```

- **inFic1:** nom logique du flux (préfixer par 'in' est une convention)
- **nom_de_fichier :** nom du fichier physique (chemin d'accès, nom, extension)
  - variable, constantes, littéral chaîne de caractère (tableau de caractère)



Exemple :

```
// Tableau de caractères équivalent à une chaîne de caractères
const char NOM_FIC[]  = "agenda.dat"; // tableau de caractères

ifstream inFic; // déclaration d'un flux de sortie

inFic.open(NOM_FIC);// ouverture du flux

if (inFic.fail())
  cerr << "erreur à l'ouverture";
  else
 {
  // premiere lecture pour vérifier si le fichier est vide
  inFic >> Vprenom >> Vnom ; // similaire à une saisie clavier
  while ( ! inFic.eof()) // tant qu'on n'est pas à la fin
  {
    // Traitement de la ligne lue
    cout << Vprenom << «  «  << Vnom<< endl ; // affichage
    // Lecture suivante
    inFic >> Vprenom >> Vnom
  }
  inFic.close() ;
```

```
 }
```

Exemple :

```
// lire un fichier mot à mot
const char NOM_FIC[]  = "unFichier.txt"; // tableau de caractères

ifstream inFic; // déclaration d'un flux de sortie

inFic.open(NOM_FIC);// ouverture du flux

if (inFic.fail())
  cerr << "erreur à l'ouverture";
  else
 {
  // une premiere lecture pour vérifier si le fichier est vide
  inFic >> Vmot ; // similaire à une saisie clavier
  while ( !inFic.eof()) // tant qu'on n'est pas à la fin
  {
    // Traitement de la ligne lue
    cout << Vmot << endl; // affichage
    // Lecture suivante
    inFic >> Vmot ;
  }
  inFic.close() ;
 }
```

III.Fichiers binaires

Alors que les fichiers textes sont constitués uniquement de caractères ASCII visibles, les fichiers binaires sont codés en fonction du type de donnée : les chaînes de caractères seront codées en ASCII, les nombres entiers en binaire, les nombres réels au format IEEE-754.
Contrairement aux fichiers texte, leur contenu n'est généralement pas lisible directement.

Les fichiers binaires sont adaptés au stockage des données structurées. Mais la longueur du type doit être fixe et connue à l'avance. Il est donc nécessaire de remplacer les types string par des tableaux de caractères.

Un **enregistrement** correspond aux données relatives à la structure gérée (un bloc de données de longueur fixe). Un fichier binaire comporte autant d'enregistrements que de structures stockées. Chaque enregistrement peut être accédé directement à partir de sa position dans le fichier.

A. Flux et fonction de gestion des fichiers binaires

flux		***actions***	
ifstream	Flux d'entrée		
ofstream	Flux de sortie		
fstream	Flux d'entrée/sortie		Selon le mode d'ouverture

		f.open(x,m)	Pour ouvrir un flux x dans un mode m : Valeur du mode : **ios::in** = en lecture seule (défaut pour ifstream) **ios::out** = en écriture (défaut par ofstream) **ios::app** = en écriture, en fin de fichier (append), préservation des données existantes **ios::nocreate** = en écriture, seulement s'il existe déjà ios::in = en lecture seule ios::in = en lecture seule ios::in = en lecture seule
		f.read(e,t)	Lecture d'un enregistrement e de taille t
		f.write(e,t)	Ecriture d'un enregistrement e de taille t
		f.seek(n,p)	Pour accéder directement à un enregistrement n à partir de la position p Valeur du positionnement : ios::beg = à partir du début
		f.close()	Pour fermer un flux
		f.eof()	Pour tester la fin d'un flux d'entrée
		f.fail()	Pour tester la bonne exécution de la dernière action sur un flux

Les données attendues par **read** et **write** doivent être converties en (char *) (convertir en un pointeur sur caractère)

B. Flux : fstream, ifstream,ostream

ATTENTION :

L'ouverture d'un flux de sortie créé le fichier s'il n'existe pas MAIS L'ECRASE S'IL EXISTE DEJA.

Syntaxe OUVERTURE/FERMETURE:

```
fstream outFic1(nom_de_fichier,mode);
```

ou bien :

```
fstream outFic1 ;
outFic1.open(("fichier.txt",mode) ;
```

- **outFic1:** nom logique attribué au flux
- **nom_de_fichier :** nom du fichier physique (chemin d'accès, nom, extension)
  - variable, constantes, littéral chaine de caractère
- **mode:** l'un des modes d'ouverture du fichier

Pour fermer le flux :

```
outFic1.close() ;
```

- **outFic1:** nom logique

Syntaxe ECRITURE:

```
outFic1.write((char*)(&enreg),longueur_type);
```

- **outFic1:** nom logique attribué au flux

- **enreg :** nom de la variable à enregistrer (&enreg = adresse de la variable)
  - variable, constantes, littéral chaine de caractère
- **longueur_type:** longueur du type de données

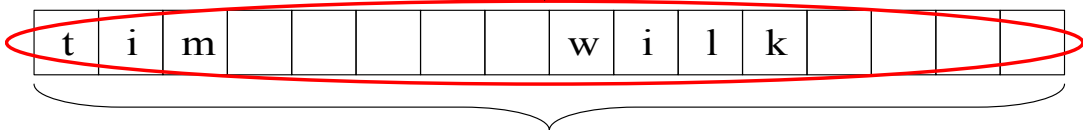
```
struct fiche {
  char Vprenom [8];
  char Vnom [8];
}
fiche enreg;

ofstream monFichier;
monFichier.open("agenda.dat",ios::out);
int Vlongueur = sizeof(fiche);

Strcpy(enreg.Vnom, "tim");
Strcpy(enreg.Vprenom, "wilk");

monFichier.write((char*)(&enreg),Vlongueur);

monFichier.close();
```



Syntaxe POSITIONNEMENT (recherche d'enregitrements):

```
outFic1.seekg(position);
```

ou bien :

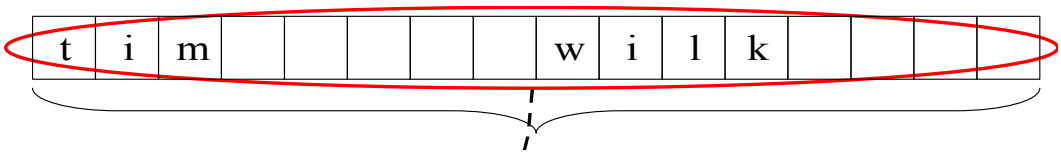
```
outFic1.seekg(deplacement,direction);
```

- **outFic1:** nom logique attribué au flux
- **position :** nombre entier (nouvelle position dans le buffer du flux)
- **déplacement:** nombre entier (déplacement dans le buffer)
- **direction :** ios::beg, ios::end, ios::cur

Syntaxe LECTURE:

```
outFic1.read((char*)(&enreg),longueur_type);
```

- **outFic1:** nom logique attribué au flux
- **enreg :** nom de la variable à enregistrer (&enreg = adresse de la variable)
  - variable, constantes, littéral chaine de caractère
- **longueur_type:** longueur du type de données (donné par **sizeof**(type))




```
struct fiche {
  char Vprenom [8];
  char Vnom [8];
}
fiche enreg;

ifstream monFichier;
monFichier.open("agenda.dat");
int Vlongueur = sizeof(fiche);

monFichier.read((char*)(&enreg),Vlongueur);

monFichier.close();
```

Exemple :

```
struct agenda
{
    char Vnom[20];
    char Vprenom[20] ;
} ;
agenda2 Vmonag;
int Vlongueur = sizeof (agenda2);

const char NOM_FIC[]  = "unFichier.txt"; // tableau de caractères

ifstream inFic; // déclaration d'un flux de sortie
inFic.open(NOM_FIC);// ouverture du flux

if (inFic.fail())
  cerr << "erreur à l'ouverture";
  else
 {
  // une premiere lecture pour vérifier si le fichier est vide
  inFic.read((char*)&Vmonag, Vlongueur);
  while ( !inFic.eof()) // tant qu'on n'est pas à la fin
  {
    // Traitement de la ligne lue
    cout << Vmonag.Vnom << Vmonag.Vprenom << endl; // affichage
    // Lecture suivante
    inFic.read((char*)&Vmonag, Vlongueur);
  }
  inFic.close() ;
 }
```

IV. Exercices – Fichiers textes

Exercice 1 :
Ecrire un programme
- qui lit les informations suivantes au clavier : référence produit, tarif
- qui écrit ces informations dans le fichier texte «catalogue.dat ».

La saisie s'arrête quand la référence contient le caractère '*'.

Exercice 2 :
Ecrire un programme qui permet la relecture du fichier précédent et en affiche les informations au fur et à mesure de la lecture.

Exercice 3 :
Ecrire un programme qui permet la relecture du fichier précédent et l'écriture dans un second fichier 'cataloguemajo.dat' comportant : la référence du produit, le tarif majoré de 10%.
Après la constitution du fichier, réafficher son contenu.

Exercice 4 :
Ecrire un programme qui permet la relecture du fichier «catalogue.dat ». et la création d'un document XML dont la DTD est la suivante :
<!ELEMENT catalogue (produit*) >
<!ELEMENT produit (reference, prix) >
<!ELEMENT reference (#PCDATA)>>
<!ELEMENT prix (#PCDATA)>

Exercice 5 :
Ecrire un programme qui permet la production du fichier 'test.cpp'dont un exemple de contenu est donné ci-dessous.
L'utilisateur va devoir saisir
- un nom du tableau,
- un nombre d'éléments dans le tableau,
- le nom d'un type de données.

Une fois le fichier 'test.cpp' créé, compilez le et vérifiez son fonctionnement.

Exemple de fichier 'test.cpp' ::
- Le nom du tableau est « **Tnombre** »
- le nombre d'éléments du tableau saisi est « **10** »
- type de donnée « **int** » :

```
// fichier généré par mon programme
#include <iostream>
using namespace std ;
int main() {
  // constantes dimension 1
  const int NB_ELEM = 10 ;
  const int IND_DEB = 0 ;
  const int IND_FIN = NB_ELEM – 1 ;

  // déclaration du tableau
  int Tnombre[NB_ELEM] ;
```

```
  // saisie
  for (int i=IND_DEB ;i<=IND_FIB ;i++)
  {
    cin >> Tnombre[i];
  }
  // affichage
  for (int i=IND_DEB ;i<=IND_FIN ;i++)
  {
    cout << Tnombre[i] << endl ;
  }
  return 0 ;
} // fin main
```