

Chapitre 6

FICHIERS BINAIRES

POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE



Les enregistrements / structures

- ☐ Un enregistrement est une structure de données formée d'un certain nombre de champs portant chacun un nom et pouvant être de types différents.
- ☐ L'enregistrement permet donc de regrouper, sous un même identificateur, des données diverses, mais logiquement interreliées.

☐

```
struct NomDuType {  
    déclaration des champs;  
};
```

```
enum Sexe { F, M };

struct Adresse { // Définition d'un type Adresse étant un enregistrement
    int    numero;
    char    rue[31], ville[41], codePostal[7];
};

struct Membre { // Définition d'un type Membre étant un enregistrement
    char        nom[51], prenom[51];
    unsigned char age;
    Sexe        sexe;
    Adresse      adresse;
    char        telephone[31];
    float        montantDu;
};

int main()
{
    // Définition de variables de type Membre
    Membre etudiant, clubSocial[50];
}
```

VI_enregistrements_membre.cpp

Les enregistrements

```
struct Point {  
    double x, y;  
};
```

 Déclaration et initialisation d'un enregistrement

```
Point point = { 12.3, -1.34 };
```

 Accès aux champs d'un enregistrement

variable.champ

```
point.x = -2.5; point.y = -6.78;
```

VI_enregistrements_point.cpp

Les enregistrements

- ✎ La seule opération agissant sur l'entité enregistrement est l'affectation entre deux enregistrements de même type.
- ✎ Toutes les autres opérations doivent être définies à l'aide des champs.
- ✎ Peuvent être utiles pour transmettre plusieurs informations en paramètre.

Traitement des éléments d'un enregistrement

```
Point pointA, pointB, vecteur[10];
```

- Initialisation d'un enregistrement

```
pointA.x = 12.3;  
pointA.y = -5.11;
```

- Affectation d'un enregistrement à un autre

```
pointB = pointA;
```

VI_enregistrements_point.cpp

Traitement des éléments d'un enregistrement

- Comparaison de deux enregistrements

```
bool pareil = false;  
if (pointA.x == pointB.x && pointA.y == pointB.y)  
    pareil = true;
```

- Le plus petit x du vecteur

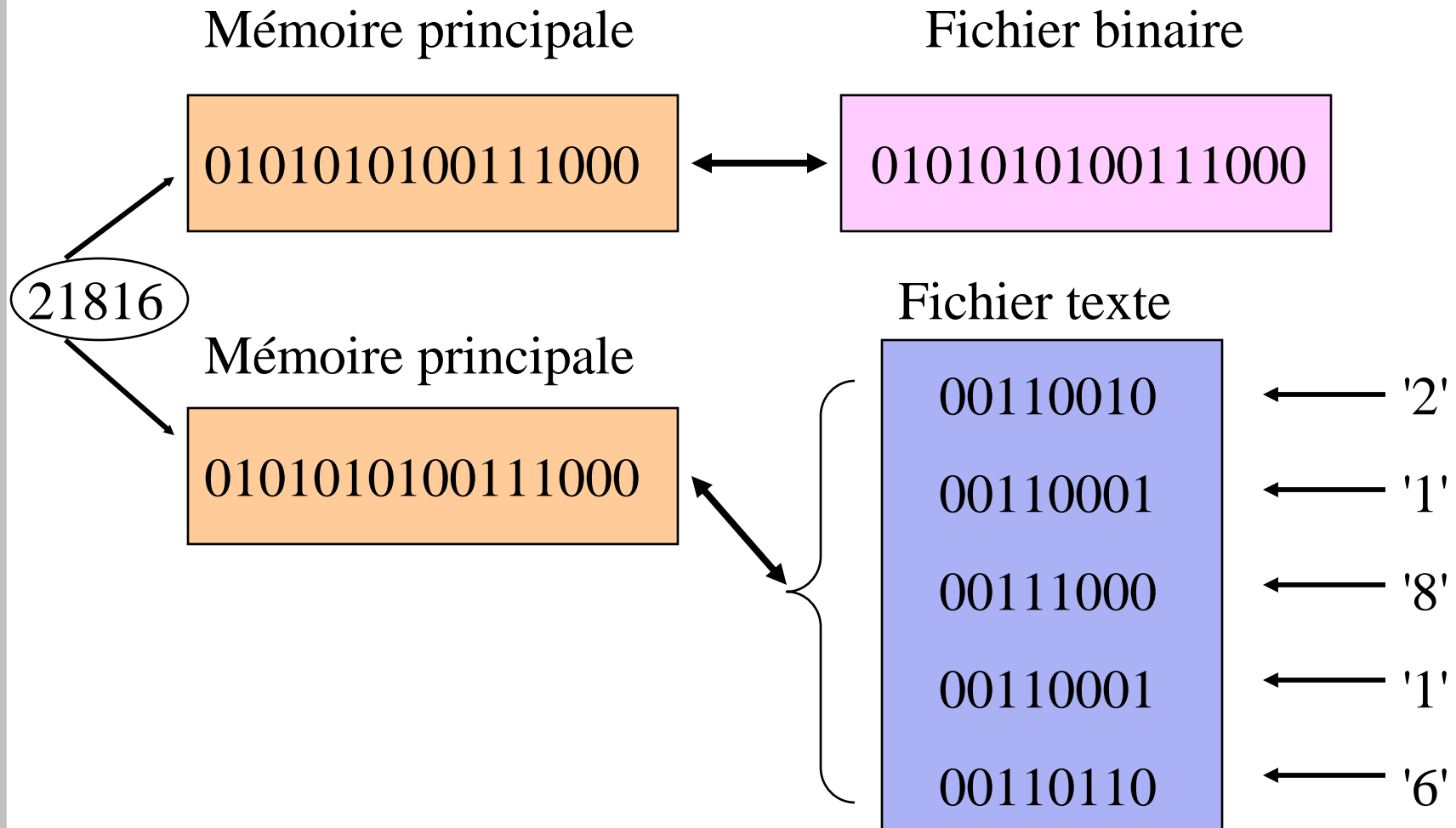
```
double minX = vecteur[0].x;  
for (int i = 1; i < 10; i++)  
    if (vecteur[i].x < minX)  
        minX = vecteur[i].x;
```

VI_enregistrements_point.cpp

Fichier binaire

- ▣ Les éléments sont représentés de la même façon que les données en mémoire.
- ▣ L'accès aux données peut être direct en se positionnant à l'emplacement désiré.
- ▣ Il n'y a pas de fin de ligne.
- ▣ L'information est sous forme codée et généralement pas intelligible lorsqu'elle est affichée avec un éditeur de texte.
- ▣ Utilise en moyenne moins d'espace.

Les fichiers binaires



Les fichiers binaires

Déclaration	<pre>ifstream f_entree; ofstream f_sortie; fstream f_InOut;</pre>
Ouverture	<pre>f_entree.open(" Lit.dat " , ios::binary); f_sortie.open(" Ecrit.dat " , ios::binary); f_InOut.open(" LitEcrit.dat " , ios::binary ios::in ios::out);</pre>
Lecture	<pre>f_entree.read((char*)&var, sizeof(var));</pre>
Écriture	<pre>f_sortie.write((char*)&var, sizeof(var));</pre>
Positionnement	<pre>f_entree.seekg(0, ios::beg); // au début du fichier f_sortie.seekg(0, ios::end); // à la fin du fichier f_entree.seekg(0, ios::cur); // à la position courante</pre>
Position	<pre>f_entree.tellg(); f_sortie.tellp();</pre>

Validation de l'ouverture

- La validation de l'ouverture du fichier a été omise dans les exemples suivants uniquement pour une question d'espace.
- La validation s'effectue à l'aide des instructions:

```
fstream fichier; // Fichier lecture et écriture

// Ouverture du fichier.
fichier.open("Fichier.ext",ios::binary|ios::in);

if (fichier.fail()) {
    cout << "Probleme d'ouverture";
}
else
    // Traitement
```

Création du fichier binaire

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

int main()
{
    // Ouverture et création du fichier, en écriture uniquement.
    ofstream fichier("entier.bin", ios::binary);

    // Inscription des valeurs 10, 20, 30, 40 et 50.
    for (int i = 1; i <= 5; i++) {
        int valeur = i * 10;
        fichier.write((char*)&valeur, sizeof(valeur));
    }
    fichier.close();
}
```

Noter qu'en C++ l'appel à 'close' sur un 'fstream' n'est pas nécessaire; le fichier est fermé automatiquement lorsque la variable du fichier n'existe plus.
Attention qu'il est nécessaire pour les 'FILE*' de C (pas vus dans ce cours).

VI_fichier_creation.cpp

Lecture du fichier binaire manière 1

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

int main()
{
    // Ouverture du fichier en lecture uniquement.
    ifstream fichier("entier.bin", ios::binary);

    // Lecture et affichage du contenu du fichier.
    while (fichier.peek() != EOF) {
        int valeur;
        fichier.read((char*)&valeur, sizeof(valeur));
        cout << valeur << endl;
    }
    fichier.close();
}
```

VI_fichier_lecture1.cpp

Lecture du fichier binaire manière 2

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

```
int main()
{
    // Ouverture du fichier en lecture uniquement.
    ifstream fichier("entier.bin", ios::binary);

    // Déterminer le nombre d'éléments dans le fichier.
    fichier.seekg(0, ios::end); // Positionnement à la fin.
    int nElements = int(fichier.tellg() / sizeof(int));

    fichier.seekg(0, ios::beg); // Repositionnement au début.
    // Lecture et affichage du contenu du fichier.
    for (int i = 0; i < nElements; i++) {
        int valeur;
        fichier.read((char*)&valeur, sizeof(valeur));
        cout << valeur << endl;
    }
    fichier.close();
}
```

'tellg()' retourne le type 'streampos'
implicitement converti en 'streamoff'
('long long' dans notre cas)

VI_fichier_lecture2.cpp

Mise à jour du fichier binaire (1)

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

int main()
{
    // Fichier en lecture et écriture.
    fstream fichier("entier.bin", ios::in|ios::out|ios::binary);

    // Modifier la troisième valeur du fichier.
    int valeur = 33;
    fichier.seekp(2*sizeof(valeur), ios::beg);    // Positionnement.
    fichier.write((char*)&valeur, sizeof(valeur)); // Écriture.
    fichier.close();
}
```

VI_fichier_miseajour1.cpp

Mise à jour du fichier binaire (2)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    // Fichier en lecture et écriture.
    fstream fichier("entier.bin", ios::in|ios::out|ios::binary);

    // Modifier la troisième valeur du fichier.
    int valeur;
    fichier.seekg(2*sizeof(valeur), ios::beg); // Positionnement.
    fichier.read((char*)&valeur, sizeof(valeur));
    cout << "La valeur actuelle est : " << valeur << endl;
    cout << "Nouvelle valeur : ";
    cin >> valeur;

    fichier.seekp(2*sizeof(valeur), ios::beg); // Repositionnement.
    fichier.write((char*)&valeur, sizeof(valeur));
    fichier.close();
}
```

VI_fichier_miseajour2.cpp

Mise à jour du fichier binaire (3)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    // Fichier en lecture et écriture.
    fstream fichier("entier.bin", ios::in|ios::out|ios::binary);

    // Modifier la troisième valeur du fichier.
    int valeur;
    fichier.seekg(2*sizeof(valeur), ios::beg); // Positionnement.
    fichier.read((char*)&valeur, sizeof(valeur));
    cout << "La valeur actuelle est : " << valeur << endl;
    cout << "Nouvelle valeur : ";
    cin >> valeur;

    fichier.seekp(-1*streamoff(sizeof(valeur)), ios::cur); // Repositionnement.
    fichier.write((char*)&valeur, sizeof(valeur));
    fichier.close();
}
```

Attention: 'sizeof' est de type 'size_t'
(qui est 'unsigned' dans notre cas)

Fonctions seekg, seekp, tellg et tellp

- C++11 supporte les « gros » fichiers
 - jusqu'à la taille maximale supportée par le système d'exploitation
- Les types sont
 - seekg/seekp prennent une position de type streamoff
 - « signed integer type sufficient for O/S maximum file size » (norme C++11 Figure 7)
 - tellg/tellp retournent un streampos
 - implicitement convertible de/vers streamoff
- Sur notre système :
 - sizeof donne un unsigned
 - streamoff est long long
 - sizeof(unsigned) < sizeof(long long)
- **Attention aux types**
 - **Mauvais:** fichier.seekg(-1*sizeof(int), ios::cur);
 - Sur notre système: -1*sizeof(int) donne 4 294 967 292.

Fichier binaire

- Les fonctions `read()`, `seekg()` et `tellg()` sont accessibles pour les fichiers de type `ifstream` et `fstream` ouvert en mode `ios::in`
- Les fonctions `write()`, `seekp()` et `tellp()` sont accessibles pour les fichiers de type `ofstream` et `fstream` ouvert en mode `ios::out`
- Les fichiers de type `fstream` ouvert en mode `ios::in` et `ios::out` ont accès à toutes les fonctions
- Les « `fstream` » ont une tête commune pour la lecture/écriture
 - Les « `stream` » généraux peuvent avoir deux têtes.
- Il est impossible d'ouvrir un fichier en mode `ios::in` s'il n'existe pas. Il peut être créé en utilisant l'une des deux possibilités suivantes:
 - `ofstream fichier1; fichier.open("sonnom.ext", ios::binary);`
 - `fstream fichier2; fichier.open("sonnom.ext", ios::binary|ios::out);`

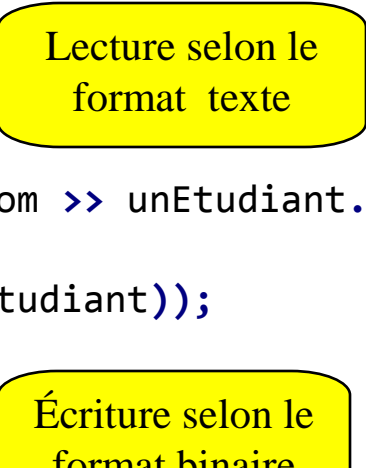
Le type string et le fichier binaire

- Il n'est pas possible dans un fichier binaire d'inscrire une donnée à l'aide d'une variable de type string. **Attention que ça peut fonctionner pour des chaînes courtes, mais il ne faut pas le faire.**
- Seulement des types qui ne contiennent pas de référence ou pointeur.
- Il faut donc utiliser la chaîne de caractères définie selon le langage C
 - `char chaine[81];`
- Pour cette raison, nous déclarons les champs de type chaîne de caractères dans un enregistrement à l'aide de sa définition en C.

```
struct Personne {  
    char nom[81], prenom[51];  
    int age;  
};
```

Fichier binaire et enregistrement

```
struct Etudiant {  
    char nom[50], prenom[50];  
    int matricule;  
};  
  
void convertirFichierTexteEnBinaire(  
    const string& nomFichierTexte,  
    const string& nomFichierBinaire)  
{  
    ifstream entree(nomFichierTexte);  
    ofstream sortie(nomFichierBinaire, ios::binary);  
  
    if (!entree.fail() && !sortie.fail()) {  
        while (!ws(entree).eof()) {  
            Etudiant unEtudiant;  
            //- Lire du fichier texte. -  
            entree >> unEtudiant.prenom >> unEtudiant.nom >> unEtudiant.matricule;  
            //- Écrire dans le fichier binaire. -  
            sortie.write((char*)&unEtudiant, sizeof(unEtudiant));  
        }  
    }  
}
```



Lecture selon le format texte

Écriture selon le format binaire

VI_fichier_et_enregistrements.cpp

Visualisation des fichiers

Texte et Binaire

Fichier Texte

```

exemple - Microsoft Visual Studio
File Edit View Project Build Debug Tools
etudiant.bin etudiant.txt fichier_etudiant.cpp
Patrick Lazure 66666
Francois Miller 1121
Julie Rioux 45127
Jean Bernier 1387
  
```

Valeur Hexadécimale
du contenu binaire

Fichier Binaire

```

exemple - Microsoft Visual Studio
File Edit View Project Build Debug Tools Test Window Help
etudiant.bin etudiant.txt fichier_etudiant.cpp binaire.cpp donnees.txt lecturebin.cpp
00000000 4C 61 7A 75 72 65 00 CC CC CC CC CC CC CC CC Lazure.....
00000010 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000020 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000030 CC CC 50 61 74 72 69 63 6B 00 CC CC CC CC CC ..Patrick.....
00000040 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000050 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000060 CC CC CC CC 6A 04 01 00 4D 69 6C 6C 65 72 00 CC ...j...Miller..
00000070 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000080 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000090 CC CC CC CC CC CC CC CC CC CC CC 46 72 61 6E 63 6F .....Franco
000000a0 69 73 00 CC CC CC CC CC CC CC CC CC CC CC CC is.....
000000b0 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
000000c0 CC CC CC CC CC CC CC CC CC CC CC CC 61 04 00 00 .....a...
000000d0 52 69 6F 75 78 00 00 CC CC CC CC CC CC CC CC Rioux.....
000000e0 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
000000f0 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000100 CC CC 4A 75 6C 69 65 00 69 73 00 CC CC CC CC CC ..Julie.is.....
00000110 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000120 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000130 CC CC CC CC 47 B0 00 00 42 65 72 6E 69 65 72 00 ...G...Bernier.
00000140 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000150 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000160 CC CC CC CC CC CC CC CC CC CC CC 4A 65 61 6E 00 00 .....Jean..
00000170 69 73 00 CC CC CC CC CC CC CC CC CC CC CC CC is.....
00000180 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
00000190 CC CC CC CC CC CC CC CC CC CC CC CC CC CC ..
000001a0
  
```

**But: créer un fichier s'il n'existe pas
et s'il existe ne pas le détruire**

Ouverture en mode in | out,
échoue si le fichier n'existe pas.

```
// - Tentative d'ouvrir un fichier existant. -
fichier.open(nomFichier, ios::binary | ios::in | ios::out);
if (fichier.fail()) {
    // - Tentative d'ouvrir un nouveau fichier vide. -
    fichier.open(nomFichier, ios::binary | ios::in | ios::out | ios::trunc);
    if (fichier.fail())
        // En cas d'échec, on indique qu'il y a eu une erreur.
        return ERREUR;
    else
        cout << "(Nouveau fichier créé.)" << endl;
        // NOTE: Normalement, cette fonction ne devrait pas afficher;
        // ici fait pour montrer quelle condition est exécutée.
}
else
    cout << "(Fichier existant ouvert.)" << endl;

// Le fichier est bien ouvert en lecture/écriture...
```

Si échoué

Avec le mode « trunc »
le fichier est créé s'il
n'existe pas,
mais effacé s'il existe.

VI_fichier_et_enregistrements.cpp

Modes d'ouverture d'un fichier

Mode	Doit exister	Efface contenu	Lire	Écrire	Notes spéciales
ios::in	■		■		
ios::out				■	
ios::out ios::trunc		■		■	
ios::out ios::app				■	Écrit toujours à la fin du fichier, les repositionnements (seekp) sont ignorés.
ios::app					
ios::in ios::out	■		■	■	
ios::in ios::out ios::trunc		■	■	■	
ios::in ios::out ios::app			■	■	Écriture remplace automatiquement la tête à la fin avant d'écrire.
ios::in ios::app					

- ios::binary et ios::ate sont indépendants de ces modes et peuvent toujours être combinés.

Lecture d'un fichier binaire

```
unsigned tailleDuFichier(istream& fichier)
{
    streampos anciennePosition = fichier.tellg(); // Conserve la position de la tête.
    fichier.seekg(0, ios::end);                  // Place la tête à la fin du fichier.
    unsigned taille = unsigned(fichier.tellg()); // La taille = position de la tête.
    fichier.seekg(anciennePosition);             // Replace la tête où elle était.
    return taille;
}
```

Fichier ouvert avant
l'appel de la fonction

```
void lireFichierBinaire(istream& fichier)
{
    unsigned nElements = tailleDuFichier(fichier) / sizeof(Etudiant);
    for (unsigned i = 0; i < nElements; i++) {
        Etudiant etudiant;
        fichier.read((char*)&etudiant, sizeof(etudiant));
        cout << etudiant.nom << ' ' << etudiant.prenom << ' '
              << etudiant.matricule << endl;
    }
}
```

VI_fichier_et_enregistrements.cpp

Ajouter un élément dans un fichier binaire

```
void ajouterEtudiant(ostream& fichier, const Etudiant& etudiant)
{
    fichier.seekp(0, ios::end); // Placer la tête à la fin du fichier.
    fichier.write((char*)&etudiant, sizeof(etudiant));
}
```

```
Etudiant demanderDonneesEtudiant()
{
    Etudiant etudiant;
    cout << "Nom: ";
    cin.getline(etudiant.nom, sizeof(etudiant.nom)-1);
    cout << "Prénom: ";
    cin.getline(etudiant.prenom, sizeof(etudiant.prenom)-1);
    cout << "Matricule: ";
    cin >> etudiant.matricule >> viderFlot;
    return etudiant;
}
```

```
void ajouterNouvelleEntreeDansFichier(ostream& fichier)
{
    Etudiant etudiant = demanderDonneesEtudiant();
    ajouterEtudiant(fichier, etudiant);
}
```

VI_fichier_et_enregistrements.cpp