

1

Lire et écrire des fichiers

420-C21-VM PROGRAMMATION II
GODEFROY BORDUAS – HIVER 2022




1

2

Les fichiers permettent une mémoire "permanente"

- ▶ Les modes textes et binaires et principaux opérateurs
- ▶ Méthode de base : ouverture et fermeture de fichier
 - ▶ Écrire dans un fichier texte
 - ▶ Lire un fichier texte
- ▶ Utiliser les fichiers pour stocker des paramètres



2

Qu'est-ce qu'un format de fichier ?

3

- ▶ Comme pour la mémoire, un fichier est composé de 0 et de 1
- ▶ Le format de fichier désigne l'encodage utilisé pour conserver les données dans le fichier
- ▶ Il existe deux formats :
 - ▶ Texte : les données sont conservées sous forme de chaîne de caractère (ASCII)
 - ▶ Binaire : les données sont conservées sous la forme numérique des nombres
- ▶ Les fichiers textes ont l'avantage des modifiables rapidement
- ▶ Les fichiers binaires ont l'avantage d'obliger l'utilisation d'une application dédiée pour les lire

3

Les principales opérations possibles

4

Le tableau suivant montre les principaux opérateurs et les principales fonctions d'entrée-sortie:

type D'OPÉRATION	Flux de données	
	écran et clavier (flux standards)	fichiers (exemple: <i>fich</i>)
écriture	<ul style="list-style-type: none"> tous les types standards : <code><<</code> exemple: <code>cout << Variable;</code> 	<ul style="list-style-type: none"> tous les types standards : <code><<</code> exemple: <code>fich << Variable;</code> pour les structures : <code>write()</code> exemple: <code>Fich.write((char*)&structure, sizeof(structure));</code>
lecture	<ul style="list-style-type: none"> tous les types standards : <code>>></code> exemple: <code>cin >> Variable;</code> le type <code>char</code> : <code>get()</code> exemple: <code>Variable = cin.get();</code> le type <code>char</code> : <code>getch()</code> et <code>_getche()</code> exemple: <code>Variable = _getche();</code> 	<ul style="list-style-type: none"> tous les types standards : <code>>></code> exemple: <code>fich >> Variable;</code> le type <code>char</code> : <code>get()</code> exemple: <code>Variable = fich.get();</code> pour les structures : <code>read()</code> exemple: <code>Fich.read((char*)&structure, sizeof(structure));</code>

Nous allons, dans un premier temps, étudier les entrées-sorties par caractère, par chaîne de caractères, et les entrées-sorties formatées (avec des valeurs numériques). Nous utiliserons ces trois formats dans le mode **texte** et dans le mode **binaire**.

4

5

Les fichiers textes

OUVRIR, LIRE ET FERMER UN FICHIER

5

6

Le type d'un fichier : *fstream*

- ▶ Nous regarderons à manipuler les fichiers en C++
 - ▶ Nous ignorons la méthode en C pur. Elle nécessite l'utilisation de pointeur
 - ▶ Pour les curieux : <https://emmanuel-delahaye.developpez.com/tutoriels/c/notes-langage-c/?page=note-de-donnees-par-un-operateur-stdin#LXXXIII>
- ▶ Le langage C++ possède un type *fstream* qui permet de manipuler les fichiers
- ▶ Il s'agit d'un flux de sortie et un flux d'entrée
 - ▶ Disons que c'est le mélange de *cout* et *cin* pour les fichiers
- ▶ Le type *fstream* est contenu dans `<fstream>`

6

Ouvrir un fichier

7

- ▶ Pour ouvrir un fichier, nous utilisons la méthode `open`

```
fstream Fichier;
Fichier.open(NomEtCheminDuFichier, ModeFichier);
```
- ▶ Les modes d'ouverture ont différents effets
 - ▶ Le choix du mode dépend de l'utilisation (lire ou écrire) un fichier
 - ▶ Le tableau à la page suivante montre les différentes ouvertures



7

Les modes de fichier

8

Mode	Type	Objectif	Effet
<code>ios::in</code>	Lecture	Ouvre pour lire le fichier	Le fichier doit exister. Si le fichier n'existe pas, <i>fail</i> est alors vrai et il n'est pas possible d'utiliser <i>fstream</i>
<code>ios::out</code>	Écriture	Ouvre pour écrire le fichier	Si le fichier n'existe pas, il sera créé. Si le fichier existe, son contenu sera effacé (perdu à jamais)
<code>ios::in ios::out</code>	Lecture et écriture	Ouvre pour lire et écrire le fichier	Le fichier doit exister. Si le fichier n'existe pas, <i>fail</i> est alors vrai et il n'est pas possible d'utiliser <i>fstream</i>
<code>ios::app</code>	Écriture	Ouvre pour écrire le fichier à la fin	Le contenu du fichier n'est pas effacé. L'écriture se fait à la fin du fichier.

8

Avant de lancer son logiciel, il faut fermer la lecture ou l'écriture

9

- ▶ Nous utilisons les fichiers « *buffisés* ». Ceci implique que l'écriture se fait uniquement à la fermeture du fichier
- ▶ Une fois terminés, nous devons libérer l'espace mémoire utilisé pour l'écriture ou la lecture
- ▶ Pour fermer un fichier, il suffit d'appeler la méthode `close`.

```
fstream Fichier;  
Fichier.close();
```

9

Ouvrir un fichier en mode écrit et écrire

10

- ▶ Pour ouvrir un fichier en écriture, nous utilisons la commande suivante :

```
fstream Fichier;  
Fichier.open("exemple_1.txt", ios::out);
```
- ▶ L'écriture dans le fichier se fait grâce à l'opérateur `<<`
- ▶ Ici, on ajoute la chaîne **Hello world**

```
Fichier << "Hello world";
```
- ▶ La commande `endl` fonctionne toujours
 - ▶ Même effet que dans `cout`

10

11

```
#include <iostream>
#include <fstream>

using namespace std;

int main(void) {
    fstream Fichier;
    Fichier.open("exemple_1.txt", ios::out);
    Fichier << "Hello world" << endl;
    Fichier << "Ligne 2";
    Fichier.close();
    return 0;
}
```

11

12

Les fichiers textes

VÉRIFIER L'OUVERTURE

12

Vérifier l'ouverture avant l'utilisation

13

- ▶ Pour de multiples raisons, il arrive que le fichier ne soit pas ouvert ou qu'une erreur se soit produite
- ▶ Il est important de vérifier
- ▶ Quand une erreur se produit, la méthode `fail` de `fstream` retourne vrai
 - ▶ Il suffit alors de vérifier avec un simple `if`

13

Fermer son application en cas d'erreur

14

- ▶ Pour terminer un programme, vous pouvez mettre fin à l'exécution de l'application grâce à la fonction `exit`
- ▶ Si l'application se termine avec une erreur, on appelle `exit` avec le paramètre `EXIT_FAILURE`
 - ▶ C'est généralement l'appel où nous l'utilisons
`exit(EXIT_FAILURE);`
- ▶ Si l'application se termine sans une erreur, on appelle `exit` avec le paramètre `EXIT_SUCCESS`
`exit(EXIT_SUCCESS);`

14

Écrire ce que l'utilisateur écrit

15

- ▶ Écrivez dans un fichier **exemple_2.txt** tout ce que la personne utilisatrice écrit dans la console.
- ▶ Utilisez un saut de ligne **vide** pour arrêter

15

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

using namespace std;

int main(void) {
    string chaine = " ";
    fstream Fichier;

    Fichier.open("exemple_2.txt", ios::out);
    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    do {
        getline(cin, chaine);
        if (chaine != "")
            Fichier << chaine << endl;
    } while (chaine != "");

    Fichier.close();

    return 0;
}
```

16

16

17

Les fichiers textes

LIRE LE CONTENU D'UN FICHIER

17

18

Lire un fichier : Un caractère à la fois

- ▶ Pour lire un fichier, il faut l'ouvrir dans le mode `ios::in`
- ▶ La méthode `get` permet de lire un caractère dans le fichier
 - ▶ Pour lire tout le fichier, il faut déplacer le curseur de lecture au caractère suivant. La méthode `get` le fait pour nous

```
char Caractere;
Caractere = Fichier.get();
```
- ▶ La méthode `eof` indique si vous avez lu le fichier jusqu'à la fin
 - ▶ La méthode retourne un `bool`

```
if (Fichier.eof())
    // Le curseur est à la fin du fichier
```

18

19

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

using namespace std;

int main(void) {
    string chaine = "";
    fstream Fichier;

    Fichier.open("exemple_3.txt", ios::in);
    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    while (!Fichier.eof())
    {
        cout << (char)Fichier.get();
    }

    Fichier.close();

    return 0;
}
```

19

20

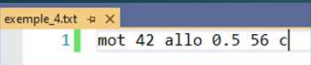
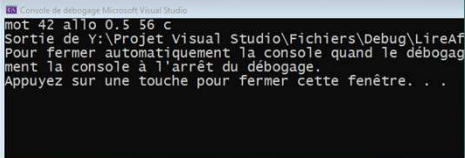
Lire un fichier : Un mot à la fois

- ▶ Comme pour `cin`, il est possible de lire un mot à la fois
 - ▶ Un mot : Ensemble de caractère séparé par un espace blanc, une tabulation ou un saut de ligne.
 - ▶ La lecture se fait grâce à l'opérateur `>>`

```
string mot;
Fichier >> mot;
```
- ▶ De plus, dans le cas d'une valeur d'un autre type que `string`, la conversion est automatique
 - ▶ Comme pour `cin`

20

21

```

#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main(void) {
    fstream Fichier;
    string mot1, mot2;
    int entier1, entier2;
    double decimal;
    char caractere;

    Fichier.open("exemple_4.txt", ios::in);
    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    Fichier >> mot1 >> entier1 >> mot2
        >> decimal >> entier2 >> caractere;

    cout << mot1 << " " << entier1 << " "
        << mot2 << " " << decimal << " "
        << entier2 << " " << caractere;

    Fichier.close();

    return 0;
}

```

21

22

```

#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main(void) {
    string chaine = "";
    fstream Fichier;

    Fichier.open("exemple_3.txt", ios::in);
    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    while (!Fichier.eof())
    {
        Fichier >> chaine;
        cout << chaine << "\n";
    }

    Fichier.close();

    return 0;
}

```

22

Lire une ligne complète

23

- ▶ La bibliothèque standard possède la fonction **getline**

```
istream& getline (istream& is, string& str, char delim);
```

 - ▶ Je vous rappelle que **istream** est la description de base d'un flux d'entrée
- ▶ La fonction **getline** permet de lire un flux jusqu'à un délimiteur précis
 - ▶ Par défaut, le saut de ligne `\n` est le délimiteur choisi
- ▶ Vous l'avez déjà vu en action en C11


```
cout << "Quel est votre nom ? ";
getline(cin, nom);

$ Quel est votre nom ? Jean Bon
```
- ▶ Le caractère de limitation n'est jamais inclus

23

24

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

using namespace std;

int main(void) {
    string chaine = "";
    fstream Fichier;

    Fichier.open("exemple_3.txt", ios::in);
    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    while (!Fichier.eof())
    {
        getline(Fichier, chaine);
        cout << chaine << "\n";
    }

    Fichier.close();

    return 0;
}
```

24

Exercice : lire un fichier source

25

- ▶ Écrivez le programme suivant :
 1. Demandez à l'utilisateur d'indiquer un chemin vers un fichier source de son choix
 2. Tant que le choix n'est pas valide (`fail() == true`), recommencer l'étape 1
 3. Affichez le contenu du fichier texte
 - ▶ Indiquez le numéro de la ligne avant d'afficher le texte
 - ▶ Vous affichez au maximum 100 caractères par ligne.
 - ▶ Vous ne pouvez pas utiliser **getline**
 4. Demander à l'utilisateur d'appuyer sur une touche de terminer le programme.

25

26

Les fichiers textes

LE FORMAT CSV

26

Exercice : créer un fichier répertoire

27

- ▶ Créer en premier lieu une structure Contact
 - ▶ Nom (string)
 - ▶ Prénom (string)
 - ▶ Date de naissance (structure)
 - ▶ Numéro de téléphone (long long)
- ▶ Utilisez une structure pour contenir la date (année/mois/jour)
- ▶ La sauvegarde est réalisé dans le format CSV (diapo suivante)
 - ▶ Nom, Prenom, Date (jour), Date (mois), Date(annee), Numéro

27

Format CSV

28

- ▶ Ancien format de donnée
- ▶ Chaque ligne correspond à une entité
- ▶ Chaque membre de l'entité est séparé par une virgule
- ▶ Exemple :

Borduas, Godefroy, 13, 12, 1919, 42

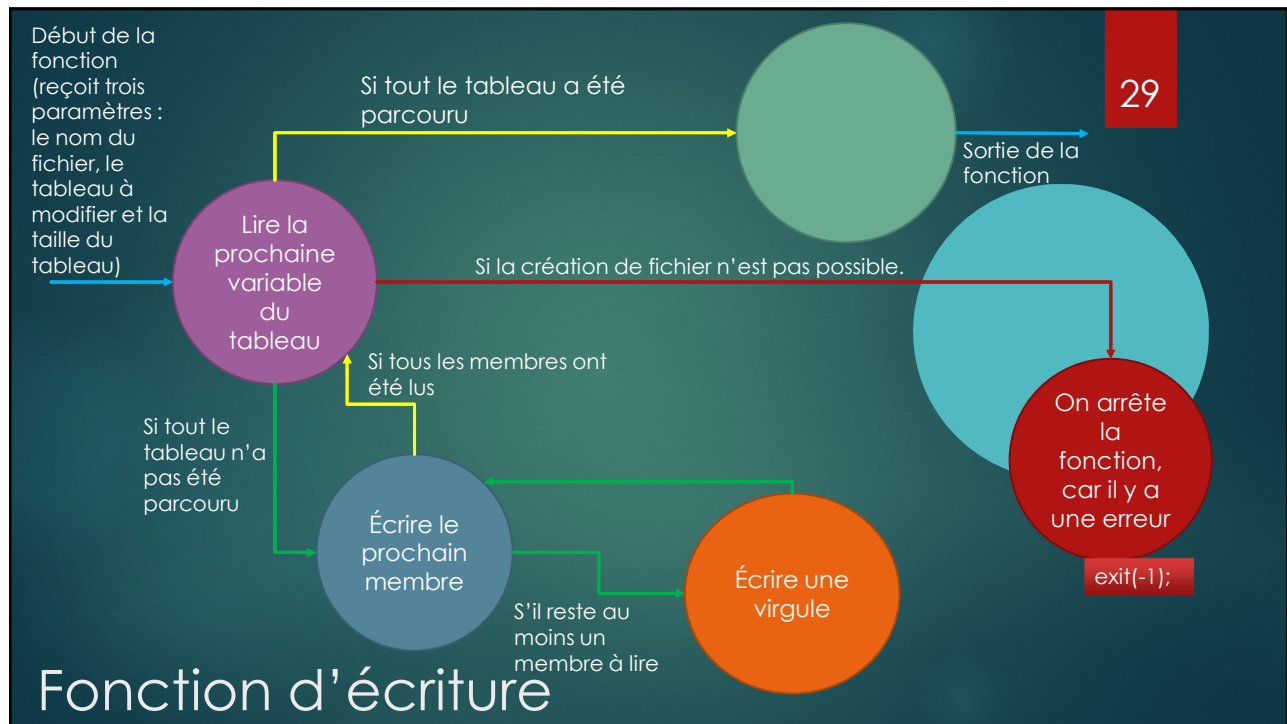
Bidule, Marilou, 14, 02, 1949, 80

Machin, Philippe, 11, 04, 1972, 70

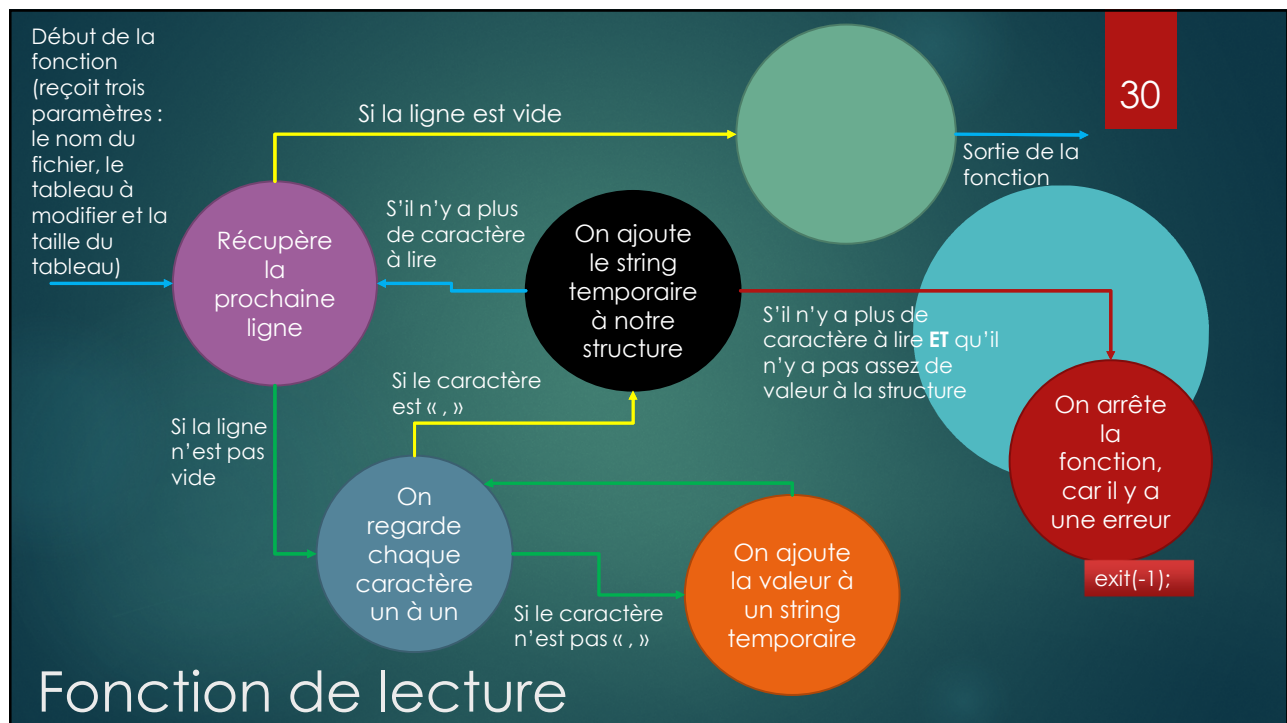
DateNaissance	Date_s
13	Jour
12	Mois
1919	Année

Personne1	Contact_s
Borduas	Nom
Godefroy	Prenom
	Date de naissance
42	Téléphone

28



29



30

31

Les fichiers binaires

OUVRIR, ÉCRIRE ET FERMER UN FICHIER

31

32

Le mode `ios::binary`

- ▶ Pour ouvrir un fichier en mode binaire, il faut spécifier le mode `ios::binary`
- ▶ Il faut aussi lui spécifier le mode d'ouverture : lecture ou écriture
 - ▶ Pour ajouter plusieurs modes, on utilise le symbole `|`
 - ▶ Lecture binaire : `fichier.open(fname, ios::in | ios::binary);`
 - ▶ Écriture binaire : `fichier.open(fname, ios::out | ios::binary);`
- ▶ Il faut toujours fermer un fichier à la fin.

32

Écrire dans un fichier

33

- ▶ Il est nécessaire d'utiliser la méthode write


```
Fichier.write((char*)&variable, sizeof(type_variable));
```
- ▶ Deux informations importantes :
 - ▶ L'adresse où trouver la variable : `&variable`
 - ▶ La conversion en `(char*)` permet de lire un octet à la fois
 - ▶ Le nombre d'octet à écrire => la taille du type de la variable
- ▶ Exemple stocker la suite de Fibonacci

33

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("./fibonacci.bin", ios::out | ios::binary);

    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    int terme1 = 1, terme2 = 1;
    Fichier.write((char*)&terme1, sizeof(int));
    Fichier.write((char*)&terme2, sizeof(int));

    while (terme2 < 75000)
    {
        int temp = terme1 + terme2;
        terme1 = terme2;
        terme2 = temp;

        Fichier.write((char*)&terme2, sizeof(int));
    }

    Fichier.close();
}
```

34

34

Version tableau

35

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("fibonacci.bin", ios::out | ios::binary);

    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    const size_t NB_TERME = 25;
    size_t fibo[NB_TERME] = {};
    fibo[0] = 1;
    fibo[1] = 1;

    for (size_t i = 2; i < NB_TERME; i++)
    {
        fibo[i] = fibo[i - 1] + fibo[i - 2];
    }

    Fichier.write((char*)&fibo, sizeof(int) * NB_TERME);
    Fichier.close();
}
```

35

36

Les fichiers binaires

LIRE UN FICHIER

36

Lire un fichier

37

- ▶ Vérifions l'état de notre fichier
- ▶ Pour lire un fichier, il faut utiliser la méthode read


```
Fichier.read((char*)&variable, sizeof(type_variable));
```
- ▶ Deux informations importantes :
 - ▶ L'adresse où ranger la valeur : &variable
 - ▶ La conversion en (char*) permet de lire un octet à la fois
 - ▶ Le nombre d'octet à écrire => la taille du type de la variable

37

```
#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("./fibonacci.bin", ios::in | ios::binary);

    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    int nbre_terme = 1, terme;

    while (!Fichier.eof()) {
        Fichier.read((char*)&terme, sizeof(int));
        cout << terme << "\t";
        if (nbre_terme % 10 == 0) {
            cout << "\n";
            nbre_terme = 1;
        }
        else
            nbre_terme++;
    }

    Fichier.close();

    cout << "\n\n";
}
```

38

Console de débogage Microsoft Visual Studio

1	1	2	3	5	8	13	21	34	55
89	144	233	377	610	987	1597	2584	4181	6765
10946	17711	28657	46368	75025	75025				

38

```

#include <iostream>
#include <fstream>

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("./fibonacci.bin", ios::in | ios::binary);

    if (Fichier.fail()) {
        cout << "Le fichier n'a pas pu ouvrir. Fin du programme";
        exit(EXIT_FAILURE);
    }

    int nbre_terme = 1, terme;

    while (!Fichier.eof()) {
        Fichier.read((char*)&terme, sizeof(int));
        cout << terme << "\t";
        if (nbre_terme % 10 == 0) {
            cout << "\n";
            nbre_terme = 1;
        }
        else
            nbre_terme++;
    }

    Fichier.close();

    cout << "\n\n";
}

```

Et on lisait des double ?

39

Il est important de lire un fichier binaire avec les types prévus.

39

Les fichiers binaires

LIRE ET ÉCRIRE DES ENREGISTREMENT

40

40

Créer un répertoire de contact

41

- ▶ Réutilisez en premier lieu une structure Contact
 - ▶ Nom (~~string~~ char[30])
 - ▶ Prénom (~~string~~ char[30])
 - ▶ Date de naissance (structure)
 - ▶ Numéro de téléphone (long long)
- ▶ Réutilisez la structure pour contenir la date (année/mois/jour)
- ▶ Pas possible d'écrire un string -> La taille varie
 - ▶ On remplace par un tableau de char

```
const int MAX = 150;

struct Date_s {
    int Annee, Mois, Jour;
};

struct Contact_s {
    char Nom[MAX], Prenom[MAX];
    Date_s DateNaissance;
    long Telephone;
};
```

41

Écrire le contact

42

- ▶ Même principe que les int -> Utiliser la méthode write


```
Fichier.write((char*)&variable_structure, sizeof(type_structure));
```
- ▶ Ouverture en append pour ajouter directement à la fin

```
void Ecrire(string fname) {
    Contact_s contact;

    cout << "Nom du contact : ";
    cin.getline(contact.Nom, MAX);
    cout << "Prenom du contact : ";
    cin.getline(contact.Prenom, MAX);
    cout << "Telephone du contact : ";
    cin >> contact.Telephone;
    cout << "Date de naissance (dd mm aaaa) : ";
    cin >> contact.DateNaissance.Jour >> contact.DateNaissance.Mois
    >> contact.DateNaissance.Annee;

    fstream Fichier;
    Fichier.open(fname, ios::app | ios::binary);
    Fichier.write((char*)&contact, sizeof(Contact_s));
    Fichier.close();
    cin.ignore(); cin.clear();
}
```

42

Afficher tous les contacts

43

- Même principe que les int -> Utiliser la méthode read

Fichier.read((char*)&variable_structure, sizeof(type_structure));

```
void AfficherTout(string fname) {
    Contact_s contact;
    int id = 1;

    fstream Fichier;
    Fichier.open(fname, ios::in | ios::binary);
    cout << left << setw(5) << "ID"
        << left << setw(MAX + 1) << "Nom"
        << left << setw(MAX + 1) << "Prenom"
        << left << setw(11) << "Telephone"
        << left << setw(10) << "Date de naissance\n";
    for (size_t i = 0; i < 2 * MAX + 35; i++)
    {
        cout << "-";
    }
}
```

```
cout << "\n";
Fichier.read((char*)&contact, sizeof(Contact_s));
while (!Fichier.eof()) {

    cout << left << setw(5) << id++
        << left << setw(MAX + 1) << contact.Nom
        << left << setw(MAX + 1) << contact.Prenom
        << left << setw(11) << contact.Telephone
        << left << contact.DateNaissance.Jour
        << "/" << contact.DateNaissance.Mois << "/"
        << contact.DateNaissance.Annee << "\n";
    Fichier.read((char*)&contact, sizeof(Contact_s));
}

Fichier.close();
```

43

Rechercher un contact par son nom

44

```
void AfficherContactNom(string fname) {
    char Nom[MAX] = {};
    Contact_s contact;
    int id = 1;
    cout << "Nom à retrouver : ";
    cin.getline(Nom, MAX);

    fstream Fichier;
    Fichier.open(fname, ios::in | ios::binary);
    cout << left << setw(5) << "ID"
        << left << setw(MAX + 1) << "Nom"
        << left << setw(MAX + 1) << "Prenom"
        << left << setw(11) << "Telephone"
        << left << setw(10) << "Date de naissance\n";
    for (size_t i = 0; i < 2 * MAX + 35; i++)
    {
        cout << "-";
    }
    cout << "\n";
```

```
Fichier.read((char*)&contact, sizeof(Contact_s));
while (!Fichier.eof()) {
    if (strcmp(contact.Nom, Nom) == 0) { // dans <string.h>
        cout << left << setw(5) << id
            << left << setw(MAX + 1) << contact.Nom
            << left << setw(MAX + 1) << contact.Prenom
            << left << setw(11) << contact.Telephone
            << left << contact.DateNaissance.Jour
            << "/" << contact.DateNaissance.Mois << "/"
            << contact.DateNaissance.Annee << "\n";
        id++;
        Fichier.read((char*)&contact, sizeof(Contact_s));
    }
}

Fichier.close();
```

44

Rechercher un contact par son numéro

45

- ▶ Il suffit de « voyager » jusqu'à la bonne position avec la méthode `seekp`

```
Fichier.seekp(nombre_octet_a_sauter, direction);
```

- ▶ Il existe trois directions possibles
 - ▶ `ios::beg` -> Depuis le début du fichier
 - ▶ `ios::cur` -> Depuis la position actuelle
 - ▶ `ios::end` -> Depuis la fin

45

46

```
void AfficherContactId(string fname) {
    Contact_s contact;
    int id;
    cout << "Numero du contact : ";
    cin >> id;

    fstream Fichier;
    Fichier.open(fname, ios::in | ios::binary);
    cout << left << setw(5) << "ID"
         << left << setw(MAX + 1) << "Nom"
         << left << setw(MAX + 1) << "Prenom"
         << left << setw(11) << "Telephone"
         << left << setw(10) << "Date de naissance\n";
    for (size_t i = 0; i < 2 * MAX + 35; i++)
    {
        cout << "-";
    }
    cout << "\n";
```

```
Fichier.seekp(sizeof(Contact_s) * (id - 1), ios::beg);
Fichier.read((char*)&contact, sizeof(Contact_s));
if (!Fichier.eof()) {
    cout << left << setw(5) << id
         << left << setw(MAX + 1) << contact.Nom
         << left << setw(MAX + 1) << contact.Prenom
         << left << setw(11) << contact.Telephone
         << left << contact.DateNaissance.Jour
         << "/" << contact.DateNaissance.Mois << "/"
         << contact.DateNaissance.Annee << "\n";
}
else
    cout << "Aucun contact a ce numero\n";
Fichier.close();
}
```

46

Comment mettre à jour ?

47

- ▶ On ouvre en lecture et écriture
 - ▶ Pour éviter de détruire le fichier
- ▶ On va à la bonne position
 - ▶ On écrit dans le fichier

