Le langage C++

Vecteurs, Pointeurs, Références, Fichiers

K. Boudjelaba

BTS SN-EC, LTP C. Carnus

Table des matières

Les vecteurs

Définitions :

- Un tableau dynamique est un tableau dont le nombre de cases (taille) peut varier au cours de l'exécution du programme (sa taille est ajustable).
- Les vecteurs (vectors) sont un type de tableaux dynamiques.

Pour utiliser les vecteurs en *C++*, il faut inclure dans l'en-tête :

```
#include <vector>
```

Pour déclarer un vecteur, il faut utiliser la syntaxe suivante :

```
vector</* type des éléments du tableau */> identifiant {};
```

Exemple :

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
    vector<double> const tableau_de_double {};
    vector<string> const tableau_de_string {};
    return 0;
}
```

La fonction **at()** :

Pour accéder à un élément d'un vecteur, on peut utiliser l'opérateur [ ]
Par exemple, pour accéder au 3ème élément du vecteur tableau_de_double, on procède comme suit :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 };
cout << vecteur[2] << endl;
```

Si on met entre les [ ], une valeur qui dépasse la "taille du vecteur -1", le programme peut planter ou avoir un comportement indéterminé.

Pour pallier ce problème, on préfère utiliser la fonction at() :

```
vector<int> vecteur = { 11, 12, 13, 14, 15 };
cout << vecteur.at(2) << endl;
```

Les fonctions front() et back() :

- La fonction **front()** permet d'accéder au premier élément
- La fonction **back()** permet d'accéder au dernier élément

Les fonctions empty() et clear() :

- La fonction **empty()** permet de savoir si le vecteur est vide ou non.
- La fonction **clear()** permet de vider tout le vecteur.

Les vecteurs - Exemples

Exemple 1 :

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> vecteur;
  // Remplissage du vecteur
  for (int i = 20; i < 29; ++i)
    vecteur.push_back(i);
  cout << "Le vecteur contient " << vecteur.size() << " éléments :"
     << endl;
  for (int i = 0; i < vecteur.size(); ++i)
  {
    cout << "\t- Valeur " << i+1 <<" est : " << vecteur[i] << endl;
  }
  // On ajoute 3 éléments
  for (int i = 0; i < 3; ++i)
  {
    vecteur.push_back(100);
  }
  cout << endl << "La nouvelle taille est : " << vecteur.size() << "
      éléments :" << endl;
  for (int i = 0; i < vecteur.size(); ++i)
  {
    cout << "\t- N-Val " << i+1 <<" est : " << vecteur[i] << endl;
  }
}
```

Les vecteurs - Exemples

Exemple 2 :

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> mon_tableau { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    // Affichage du 1er et dernier élément
    cout << "Le premier élément du tableau : " << mon_tableau.front
     () << "." << endl;
    cout << "Le dernier élément du tableau : " << mon_tableau.back()
      << "." << endl;
    // Ajout d'éléments au tableau
    mon_tableau.push_back(11);
    mon_tableau.push_back(12);
    // Modification du 2ème élément.
    mon_tableau[1] = 20;
    // Affichage de la taille du nouveau tableau
    cout << "Taille du nouveau tableau : " << mon_tableau.size() <<
     endl;
    // Retirer le dernier élément
    mon_tableau.pop_back();
    cout << "Affichage des éléments du nouveau tableau:\n";
    for (int element : mon_tableau)
    {
        cout << element << endl;
    }
/* ---------- Suite : page suivante ---------- */
```

Exemple 2 (suite) :

```
/* ---------- Suite : ---------- */
    // Vider le tableau.
    mon_tableau.clear();
    cout << "Taille après avoir vidé le tableau : " << mon_tableau.
     size() << endl;

    // Remplacement du tableau par un tableau de taille 4, en
     assignant 4 fois la valeur 33
    mon_tableau.assign(4, 33);
    cout << "Taille actuelle de mon tableau : " << mon_tableau.size
     () << "\n";
    cout << "Les valeurs actuelles sont : " << "\n";
    for (int valeur : mon_tableau)
    {
        cout << valeur << endl;
    }
    cout << '\n';

    return 0;
}
```

Les pointeurs

Case mémoire et adresse

- Tout objet (variable, fonction . . .) manipulé par l'ordinateur est stocké dans sa mémoire, constituée d'une série de cases.
- Pour accéder à un objet (au contenu de la case mémoire dans laquelle cet objet est enregistré), il faut connaître le numéro de cette case. Ce numéro est appelé l'adresse de la case mémoire.
- Lorsqu'on utilise une variable ou une fonction, le compilateur utilise l'adresse de cette dernière pour y accéder.

Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable (objet).

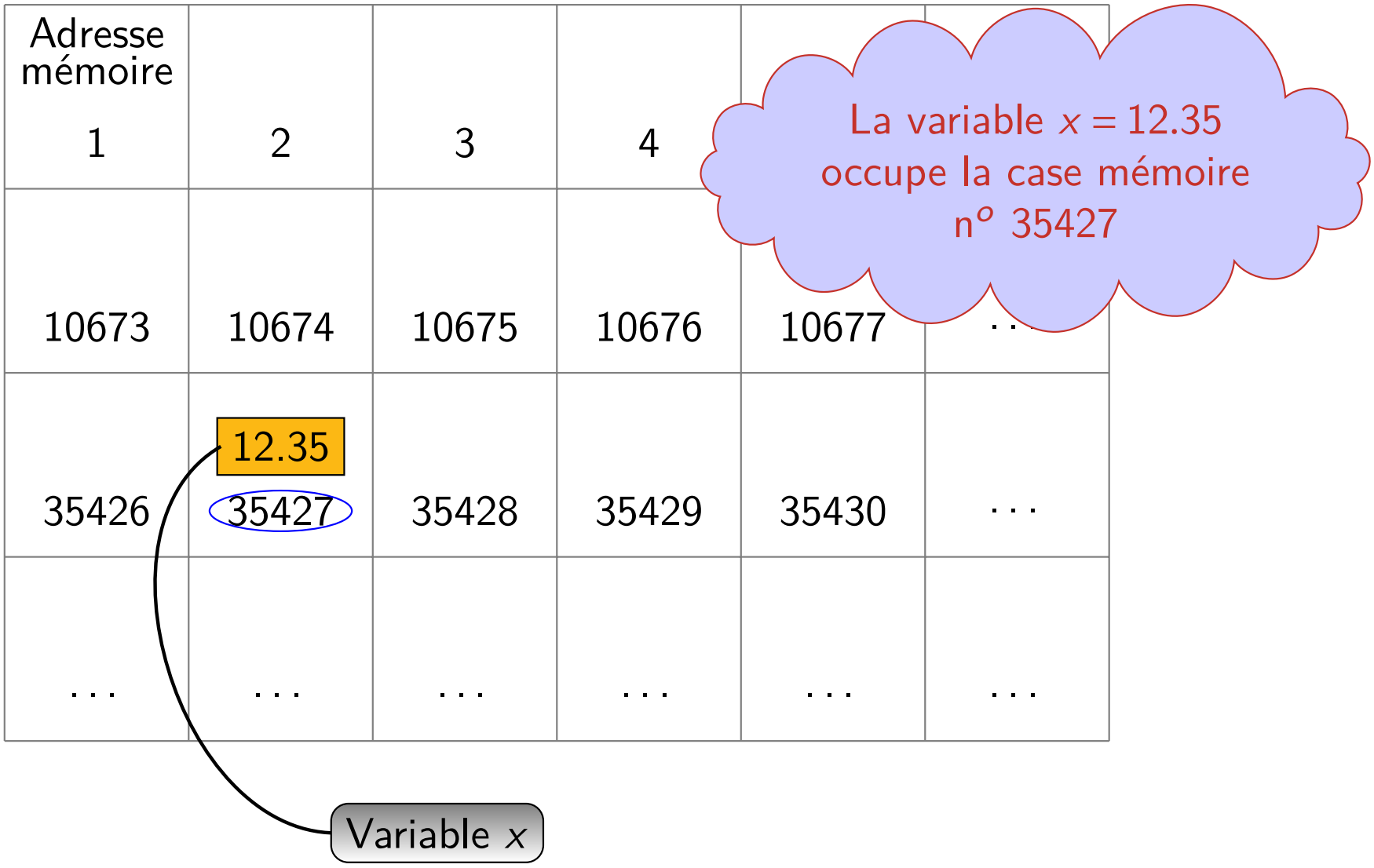
Les pointeurs

Adresse mémoire					

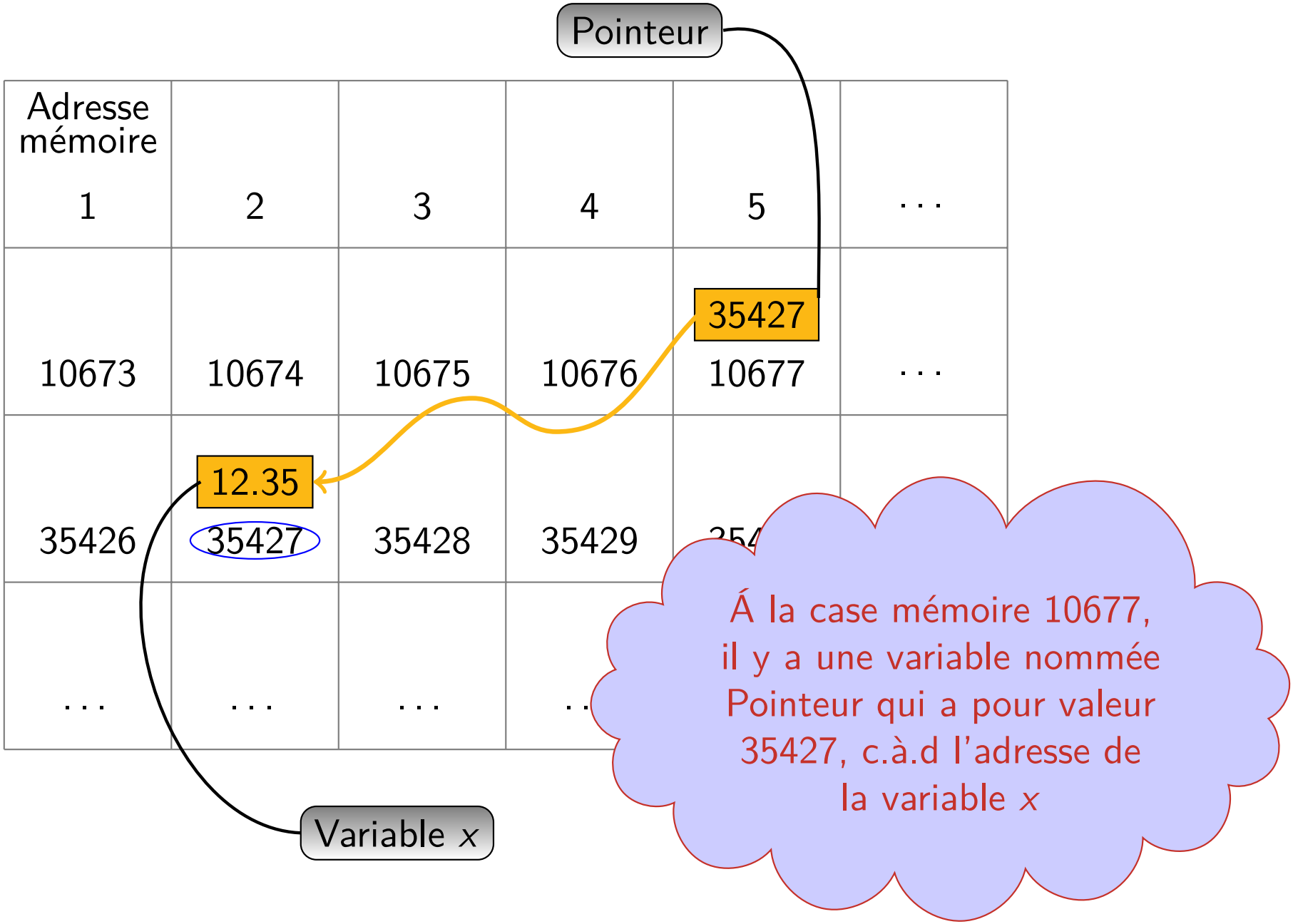
Les pointeurs

Adresse mémoire 1	2	3	4	5	···
10673	10674	10675	10676	10677	···
35426	35427	35428	35429	35430	···
···	···	···	···	···	···

Les pointeurs



Les pointeurs



Les pointeurs

Pointeur :

Pour déclarer un pointeur, on procède de la même manière que pour les variables, c.à.d. déclarer :

- le type
- le nom, précédé par $*$

```
int *pointeur;
```

On peut aussi utiliser cette syntaxe :

```
int* pointeur;
```

Inconvenient : ne permet pas de déclarer plusieurs pointeurs sur la même ligne.

```
int* p1, p2, p3, p4;
```

Seul $p1$ sera un pointeur, les autres variables seront des entiers standards

Exemple :

```
string *pointeur1;
//Un pointeur qui peut contenir l'adresse d'une chaîne de caractères

vector<int> *pointeur2;
//Un pointeur qui peut contenir l'adresse d'un tableau de nombres
    entiers
```

Les pointeurs

Exemple 1 :

```
#include <iostream>
using namespace std;
int main()
{
    int x = 10;
    cout << &x << endl;
    return 0;
}
```

En C++, le symbole pour obtenir l'adresse d'une variable est &. Pour afficher l'adresse de la variable x, on doit écrire $\&x$.
Après exécution du code, la console affiche $0x7ffeefbff538$, qui correspond à l'adresse (en hexadécimal) de la case mémoire contenant la variable x.

Exemple 1 (suite) :

```
#include <iostream>
using namespace std;
int main()
{
int x = 10;
cout << &x << endl;

int *p_x = nullptr
// ou int *p_x(0);
p_x = &x;
cout << p_x << endl;
return 0;
}
```

Important : Les pointeurs doivent être déclarés en les initialisant à 0.
La console affiche :
$0x7ffeefbff538$
$0x7ffeefbff538$
L'adresse de x est sauvegardée dans le pointeur p_x.
On dit alors que le pointeur p_x pointe sur x.

Les pointeurs

Exemple 2 :

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

int main()
{
int a, b, c;
int *x(nullptr), *y(nullptr);

a = 98;
x = &a;
c = *x + 5;
y = &b;
*y = a + 10;

cout << "b = " << b << endl;
cout << "c = " << c << endl;

return 0;
}
```

La console affiche :

b = 108

c = 103

- a est initialisé à 98.
- $x = \&a$; affecté à x l'adresse de a. x est un pointeur vers a.
- $*x$ est la variable pointée par x, c.à.d a (=98).
- $c = *x + 5$; permet de transférer $98+5=103$ dans la variable c.
- $y = \&b$; permet de mettre dans la variable y l'adresse de la variable b. y un pointeur vers b. $a+10=98+10=108$.
- $*y = a + 10$; permet de transférer dans la variable pointée par y la valeur de $a + 10 = 108$. On stocke 108 dans b de manière indirecte via le pointeur y.
- Affichage des valeurs de b et c.

Les références

Référence

- Une référence peut être vue comme un alias d'une variable (utiliser la variable ou une référence à cette variable est équivalent).
- On peut modifier le contenu de la variable en utilisant une référence.
- Une référence ne peut être initialisée qu'une seule fois : à la déclaration. Toute autre affectation modifie en fait la variable référencée.
- Une référence ne peut donc référencer qu'une seule variable.
- Les références sont principalement utilisées pour passer des paramètres aux fonctions.

Déclaration

```
type &reference = identificateur;
```

Exemple

```
int x = 0;
int &r_x = x;          // Référence sur la variable x.
r_x = r_x + x;         // Double la valeur de x (et de r_x).
```

Les références

Exemple 1 :

```
#include <iostream>
using namespace std;
int main()
{
    int a = 98, b = 78, c;
    int &x = a;
    c = x + 5;  // équivalent à :  c = a + 5;
    int &y = b;
    y = a + 10; // équivalent à :  b = a + 10;
    cout << "b = " << b << endl;
    cout << "c = " << c << endl;
    return 0;
}
```

La console affiche :

```
b = 108
c = 103
```

- `int &x = a;` permet de déclarer une référence x vers la variable a.
- `c = x + 5;` permet donc de transférer 103 dans la variable c.
- `int &y = b;` permet de déclarer une référence y vers la variable b.
- `y = a + 10;` permet de transférer 108 dans la variable b.

Les fichiers

Problématique

Les variables manipulées dans un programme sont des adresses de mémoire vive (RAM) et disparaissent à chaque fin d'exécution du programme ou au plus tard dès la fermeture du programme.
Les fichiers servent donc à stocker des informations de manière permanente.

Définitions

- Un fichier est un ensemble de données numériques réunies sous un même nom, enregistrées sur un support de stockage.
- Dans un nom de fichier, on trouve souvent l'extension , qui renseigne sur la nature des informations contenues dans le fichier et les logiciels utilisables pour le manipuler.
- Un fichier contient des métadonnées : son auteur, sa date de création, les personnes autorisées à le manipuler . . .

Les fichiers

Afin de faciliter leur localisation, les fichiers sont disposés et organisés dans des systèmes de fichiers qui permettent de placer les fichiers dans des emplacements appelés répertoires (dossiers).

→ Les fichiers sont répartis dans une arborescence de dossiers et on peut les localiser à partir d'un chemin d'accès.

Un chemin d'accès permet d'indiquer l'emplacement d'un fichier dans le système de fichiers. Il contient le nom du fichier concerné, mais également, un ou plusieurs noms de dossiers, qu'il est nécessaire de traverser pour accéder au fichier depuis la racine.

Exemples :

- Chemin absolu : On peut utiliser / à la place de \

```
C:\Documents\Cours_Informatique\Data.txt
```

- Chemin relatif : si on est déjà dans le dossier Documents

```
\Cours_Informatique\Data.txt
Ou
.\Cours_Informatique\Data.txt
```

Types d'accès

Le type d'accès est la manière dont la machine procède pour aller rechercher les informations contenues dans le fichier.

- **L'accès séquentiel :** on ne peut accéder qu'à la donnée suivant celle qu'on vient de lire. On ne peut donc accéder à une information qu'en ayant au préalable examiné celle qui la précède.
- **L'accès direct :** on peut accéder directement à l'enregistrement de son choix, en précisant le numéro de cet enregistrement.
- **L'accès indexé :** il combine la rapidité de l'accès direct et la simplicité de l'accès séquentiel. Il est adapté au traitement des gros fichiers (bases de données par exemple).

Modes d'ouverture

- **Ouverture en mode lecture :** On peut uniquement récupérer les informations qu'il contient, sans pouvoir les modifier
- **Ouverture en mode écriture :** On peut écrire toutes les informations que l'on veut. Mais les informations précédentes, si elles existent, seront intégralement écrasées.
- **Ouverture en mode ajout :** on ne peut ni lire, ni modifier les informations existantes. Mais on peut ajouter de nouvelles informations.

Les fichiers

Les fichiers textes :

Ils contiennent des informations sous la forme de caractères (en utilisant le code ASCII). Exemple : Le code source d'un programme C++

Les fichiers binaires :

Ils contiennent directement la représentation mémoire des informations. Un fichier binaire peut aussi être vu comme une séquence d'octets. Exemple : le résultat de la compilation d'un programme C++ (programme exécutable)

Les fichiers

Écrire dans un fichier :

- L'ouverture et l'écriture d'un fichier se fait avec l'objet **ofstream**
- Les modes d'ouverture sont :
  - **ios::app** (append) ajout à la fin du fichier
  - **ios::ate** (at end) met l'index à la fin du fichier
  - **ios::binary** pour ouvrir un fichier binaire
  - **ios::in** (input) permet la lecture
  - **ios::out** permet l'écriture
  - **ios::trunc** (truncate) vide le fichier à l'ouverture

Remarque : **ofstream** est par défaut un mode d'ouverture **ios_base::out | ios_base::trunc** (ouverture en écriture et effacement du contenu du fichier)

Les fichiers

Il est préférable, avant toute opération sur un fichier, de tester les indicateurs d'état en utilisant une des méthodes suivantes :

- **bad()** : Retourne la valeur True si une opération de lecture échoue. Exemple : essayer d'écrire dans un fichier qui n'est pas ouvert en écriture
- **fail()** : Retourne la valeur True dans le même cas que **bad()** et aussi dans le cas où une erreur de format se produit.
- **eof()** : Retourne la valeur True si un fichier ouvert en lecture a atteint la fin
- **good()** : Retourne la valeur False lorsque les méthodes précédentes retourneraient True

Exemple :

```
ofstream fichier {Data.txt}; // Ou ofstream fichier (Data.txt);
if (fichier.good())
{
fichier << 180 << endl; // Ecriture
}
else
{
cerr << "ERREUR : Impossible d'écrire dans le fichier !" << endl;
exit(1);
}
```

En C++, on peut utiliser la librairie fstream (file stream).

```
#include <fstream>
```

- **Ouverture de fichiers :**

```
ofstream fichier { "Data.txt" };
// Ou
ifstream fichier { "Data.txt" };
// A utiliser quand le fichier existe
```

On ouvre un fichier nommé "Data.txt", qui se trouve dans le dossier du projet. S'il existe, il sera ouvert. Sinon, il sera d'abord créé puis ouvert.

- **Ecriture dans un fichier :**

```
ofstream fichier { "Data.txt" };
// On écrit un 5, une tabulation et un 10.
fichier << 5 << "\t" << 10;
```

- **Ouverture sans effacer (ouverture en mode ajout) :**

```
std::ofstream fichier { "Data.txt", std::ios::app };
```

app : append (ajouter).

Exemples : (voir les pages suivantes)

- Écrire le code le l'exemple 1
- Exécuter le programme
- Noter ce qui est affiché sur la console après l'exécution
- Ouvrir le fichier "Test.txt" après l'exécution et vérifier le contenu du fichier
- Conclure
- Refaire les mêmes étapes pour les exemples 2, 3, 4, 5 et 6.

Exemple 1 :

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

int main()
{
    ofstream fichier { "C:/Documents/Test.txt"};
    /*   Ou
    string const nomFichier { R"(C:/Documents/Test.txt)" };
    ofstream fichier { nomFichier };
    */

    fichier << 2021;

    string texte1 { u8"Lycée Charles Carnus." };
    fichier << "\t" << texte1;

    fichier << "\n" << 12;

    cout << "Fin du programme" << endl;

    return 0;
}
```

Exemple 2 :

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

int main()
{
    ofstream fichier { "C:/Documents/Test.txt", std::ios::app };

    string texte2 { u8"BTS SN-EC." };
    fichier << "\t" << texte2;

    int x = 50;
    fichier << "\n" << x << "+ 10 = " <<60;

    cout << "Fin du programme" << endl;

    return 0;
}
```

Exemple 3 :

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

int main()
{
    ifstream fichier { "C:/Documents/Test.txt" };

    int entier { 0 };
    fichier >> entier;
    cout << "Le nombre entier vaut : " << entier << endl;

    string phrase { "" };
    fichier >> phrase;
    cout << "La phrase est : " << phrase << endl;

    int nombre { 0 };
    fichier >> nombre;
    cout << "Le deuxième nombre : " << nombre << endl;

    return 0;
}
```

Exemple 4 :

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

int main()
{
    ifstream fichier { "C:/Documents/Test.txt" };

    int entier { 0 };
    fichier >> entier;
    cout << "Le nombre entier vaut : " << entier << endl;

    string phrase { "" };
    getline(fichier, phrase);
    cout << "La phrase est : " << phrase << endl;

    int nombre { 0 };
    fichier >> nombre;
    cout << "Le deuxième nombre : " << nombre << endl;

    return 0;
}
```

Exemple 5 :

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

int main()
{
    ifstream fichier { "C:/Documents/Test.txt" };

    int entier { 0 };
    fichier >> entier;
    std::cout << "Le nombre entier vaut : " << entier << endl;

    string phrase { "" };
    getline(fichier >> std::ws, phrase);
    cout << "La phrase est : " << phrase << endl;

    int nombre { 0 };
    fichier >> nombre;
    cout << "Le deuxième nombre : " << nombre << endl;

    return 0;
}
```

Exemple 6 :

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

int main()
{
    ifstream fichier { "C:/Documents/Test.txt" };
    string ligne { "" };

    while (getline(fichier, ligne))
    {
        cout << "Ligne lue : " << ligne << endl;
    }

    return 0;
}
```

Exemple 7 :

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

int main()
{
    ofstream fout;
    fout.open("Sortie.txt");
    fout<<"Mon fichier text\n";

    int nombre = 10;
    fout<<"Le chiffre est : "<<nombre<<endl;
    fout<<"Fin fichier";
    fout.close();

    return 0;
}
```

La manipulation d'un fichier binaire s'effectue presque comme celle d'un fichier texte. La différence réside au niveau de la désignation du fichier et l'écriture (la lecture) des données.

$\ll \rightarrow$ **f.write()**

$\gg \rightarrow$ **f.read()**

Pour créer un fichier binaire, il faut ajouter le mode **ios::binary**

ofstream fichier("Data.txt", **ios::binary**);

Pour écrire une variable y ($\forall$ son type) : fichier.write((char*)x, sizeof(x));
Pour lire une variable y à partir d'un fichier ($\forall$ son type) :
fichier.read((char*)x, sizeof(x));

Curseur dans le fichier

- **fichier.tellp();** : renvoie la position de la tête d'écriture (de lecture), exprimée en nombre d'octets depuis le début du fichier.

```
ofstream fichier("Data.txt");
int position  = fichier.tellp();   //On récupère la position
cout<< "Nous nous situons  au " << position  << "ème caractère du
    fichier." <<endl;
```

Curseur dans le fichier

- **fichier.seekp(nombre d'octets, position);** : permet de se déplacer. Les trois positions possibles sont :
  - Début du fichier : **ios::beg**
  - Fin du fichier : **ios::end**
  - Position actuelle : **ios::cur**

```
ifstream fichier("Data.txt");
fichier.seekg(10,ios::beg); //se placer au 10ème caractère après le
    début du fichier
fichier.seekg(20,ios::cur); //aller 20 caractères plus loin que l'
    endroit où se situe le curseur
```

La taille

Pour connaitre la taille d'un fichier, on se déplace à la fin et on demande au curseur où il se trouve.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
ifstream fichier("Data.txt"); //On ouvre le fichier
fichier.seekg(0, ios::end);//On se déplace à la fin du fichier
int taille;  taille = fichier.tellg(); //On récupère la position qui
     correspond donc à la taille du fichier !
cout << "Taille du fichier :" << taille << " octets." << endl;
return 0; }
```

Définitions :

- Un fichier CSV (Comma-Separated Values: Valeurs séparées par des virgules) désigne un fichier informatique de type tableur, dont les valeurs sont séparées par des virgules.
  Un fichier CSV est un fichier texte, par opposition aux formats binaires.
- Chaque ligne du texte correspond à une ligne du tableau et les virgules correspondent aux séparations entre les colonnes. Les portions de texte séparées par une virgule correspondent ainsi aux contenus des cellules du tableau. Une ligne est une suite ordonnée de caractères terminée par un caractère de fin de ligne.
- Selon le logiciel et les paramètres, on peut parfois utiliser un point-virgule, un espace ou d'autres caractères, pour séparer les différentes données d'une même ligne.

Exemple : Création d'un fichier CSV

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

int main() {
    ofstream fichier("C:/Documents/Tableur1.csv");

    fichier << "En-Tête1, En-Tête2\n";
    fichier << "10, 11\n";
    fichier << "20, 21\n";
    fichier << "30, 31\n";

    fichier.close();

    return 0;
}
```

Exemple : Création d'un fichier CSV

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

using namespace std;

int main()
{
    ofstream myFile;
    myFile.open("test.csv");

    for (int i=0; i<10; i++)
    {
        myFile<<i<<","<<i*i<<endl;
    }
    myFile.close();
    cout<<"Fin programme"<<endl;
    return 0;
}
```

Exemple : CSV – Tableau avec 1 colonne

```cpp
#include <string>
#include <fstream>
#include <vector>
using namespace std;

void ecrire_csv(string filename, string colname, vector<int> vals)
{
    // Fonction pour écrire dans un fichier CSV
        // filename :   Nom du fichier
        // colname :    Nom de l'unique colonne
        // vals :       Vecteur de valeurs entières
    ofstream fichier(filename);
    fichier << colname << "\n";
    for(int i = 0; i < vals.size(); ++i)
    {
        fichier << vals.at(i) << "\n";
    }
    fichier.close();
}

int main()
{
    vector<int> vec(10, 1);
    ecrire_csv("C:/Documents/Tableur2.csv", "Colonne 1", vec);
    return 0;
}
```

Exemple : CSV – Tableau avec 3 colonnes

```
#include <string>
#include <fstream>
#include <vector>
#include <utility>  //pair
using namespace std;

void write_csv(string filename, vector<pair<string, vector<int>>>
     dataset){
    // Chaque colonne est représentée par pair <nom col, données>
    // selon pair<string, vector<int>>

    ofstream fichier(filename);

    for(int j = 0; j < dataset.size(); ++j)
    {
        fichier << dataset.at(j).first;
        if(j != dataset.size() - 1) fichier << ",";
    }
    fichier << "\n";

    for(int i = 0; i < dataset.at(0).second.size(); ++i)
    {
        for(int j = 0; j < dataset.size(); ++j)
        {
            fichier << dataset.at(j).second.at(i);
            if(j != dataset.size() - 1) fichier << ",";
        }
```

Exemple : CSV – Tableau avec 3 colonnes (suite)

```
            fichier << "\n";
        }

        fichier.close();
}

int main() {
        vector<int> vec1(10, 1);
        vector<int> vec2(10, 2);
        vector<int> vec3(10, 3);

        vector<pair<string, vector<int>>> vals = {{"Colonne1", vec1}, {"
         Colonne2", vec2}, {"Colonne3", vec3}};

        write_csv("C:/Documents/Tableur3.csv", vals);
        return 0;
}
```

Exemple : Lecture d'un fichier CSV

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

using namespace std;

int main()
{
    ifstream myFile;
    myFile.open("test.csv");

    while (myFile.good())
    {
        string line;
        getline(myFile, line, ',');
        cout<< line << endl;
    }
    myFile.close();
}
```

Exemple : Lecture et duplication d'un fichier CSV

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

using namespace std;

int main()
{
    ifstream ifs( "C:/Documents/Tableur3.csv" );
    ofstream ofs( "C:/Documents/Tableur4.csv" );
    string ligne;
    while ( getline( ifs , ligne ) )
    {
        ofs << ligne << endl;
        cout << "Ligne lue : " << ligne << endl;
    }
    return 0;
}
```

Exercice 1 :

Ecrire une fonction qui a comme paramètres un tableau d'entiers de taille quelconque, et 2 pointeurs vers des entiers min et max. La fonction doit renvoyer dans les entiers pointés par min et max respectivement les plus petits et les plus grands entiers du tableau.

Exercice 2 :

Écrire une fonction pour calculer la factorielle d'un nombre entier. Sa déclaration sera `int fact(int &n)`.

$n! = 1 \times 2 \times 3 \cdots \times (n-1) \times n$

Exercice 3 :

Écrire un programme pour saisir les notes puis calculer la moyenne. L'utilisateur peut entrer autant de valeurs qu'il souhaite, cependant, un nombre négatif signifiera qu'il ne souhaite plus entrer de nouvelles notes.

Exercice 4 :

- Écrire un programme qui permet de saisir et de mémoriser dans un fichier nommé **Contact.txt**, le nom et le numéro de téléphone de 5 personnes.
- Écrire un programme pour Afficher le contenu du fichier Contact.txt.
- Écrire un programme qui permet d'afficher à l'écran le contenu du fichier **Contact.txt**, en sautant une ligne entre chaque personne.

- La fonction qui permet de lire un fichier texte ligne par ligne est : getline()
- **Syntaxe :** bool getline(ifstream , string line)
- getline() renvoie un bool indiquant si l'on peut continuer à lire :
  - True : il reste encore des données
  - False : c'est la fin du fichier

N.B. Voir l'exemple 6 en page 31 pour plus de détails.

Exercice 5 :

- Écrire un programme qui permet de saisir et de mémoriser dans un fichier binaire nommé **Contact_bin.txt**, le nom et le numéro de téléphone de 5 personnes.
- Écrire un programme qui permet d'afficher à l'écran le contenu du fichier binaire **Contact_bin.txt**, en sautant une ligne entre chaque personne.

Exercice 6 :

Écrire un programme pour créer le fichier CSV représenté dans la figure suivante :

1	Etudiant1	12	13	14	11
2	Etudiant2	15	10	12	14
3	Etudinat3	10	9	15	12
4	Etudiant4	16	10	9	14
5	Etudiant5	13	8	15	16
6	Etudiant6	15	12	14	10
7	Etudiant7	15	15	11	19
8	Etudinat8	13	11	13	12
9	Etudiant9	14	12	10	14
10	Etudiant10	12	12	14	13

Exercice 7 :

Ecrire un programme qui ouvre un fichier et récupérer les informations suivante :

- Le nombre de lignes du fichier
- Le nombre de caractères (sans les espaces)
- Le nombre de mots