CHARLES CARNUS
LYCÉE TECHNIQUE
ENSEIGNEMENT SUPÉRIEUR

Programmation

C++ : Les fichiers

Prof : Kamal Boudjelaba

15 mars 2024

B T S S N

LTP Charles Carnus
Avenue de Saint Pierre - Bourran - 12000 RODEZ, France

IR–EC

Table des matières

Liste des tableaux

1. 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.
Pour permettre l'échange de données entre des applications, il est nécessaire de stocker les données dans un format clairement spécifié.
La lecture et l'écriture de fichiers selon une spécification donnée joue donc un rôle clé dans les applications de calcul scientifique.

Définition 1.1 : Les fichiers

- 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 . . .

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.

Chemins d'accès aux fichiers

— 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`

`.\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.

Définition 1.2 : 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++

Définition 1.3 : 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)

Ecrire dans un fichier

— L'ouverture et l'écriture sur 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 1.1 : Ouverture de fichiers

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

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 1.1

```
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).

1.1 Ecriture dans un fichier

Exemple 1.2

```
1  #include <cassert>
2  #include <iostream>
3  #include <fstream>
4
5  int main(int argc, char* argv[])
6  {
7      double x[3] = {0.0, 1.0, 0.0};
8      double y[3] = {0.0, 0.0, 1.0};
9      std::ofstream write_output("Output.dat");
10     assert(write_output.is_open());
11     for (int i=0; i<3; i++)
12     {
13         write_output << x[i] << " " << y[i] << "\n";
14     }
15     write_output.close();
16     return 0;
17 }
```

L'écriture ou la lecture d'un fichier nécessite le fichier d'en-tête supplémentaire `fstream`.
Ce code montre comment écrire dans un fichier.
On déclare d'abord une variable de flux de sortie `write_output` en la spécifiant comme étant de type std::ofstream, et on spécifie également le nom de fichier `"Output.dat"` comme indiqué à la ligne 9.
La ligne 10 vérifie ensuite que le fichier a été ouvert avec succès.
L'écriture dans le fichier est similaire à la sortie de la console, mais en remplaçant `std::cout` par write_output à la ligne 13 : cela écrit les entrées des tableaux `x` et `y` dans le fichier associé à la variable de flux de sortie, dans ce cas `Output.dat`.
Enfin, à la ligne 15, lorsque toutes les données requises ont été écrites dans le fichier, on "ferme le descripteur de fichier".

Note 1.1. La sortie de la console est mise en mémoire tampon, et donc la sortie peut ne pas être immédiatement écrite sur la console. La sortie vers le fichier est également mise en mémoire tampon : la fermeture du descripteur de fichier vide la mémoire tampon : c'est-à-dire que toutes les données qui ont été mises en mémoire tampon sont écrites dans le fichier avant que l'ordinateur n'exécute d'autres instructions. Il est important que cela soit fait : si une autre partie du programme lit un fichier qui est encore en cours d'écriture, alors on ne peut pas être certains des données, le cas échéant, qui ont déjà été écrites sur le disque. La fermeture du descripteur de fichier a pour effet supplémentaire qu'aucune autre donnée ne peut être écrite dans ce fichier : cela empêche le fichier d'être corrompu en essayant par erreur d'écrire d'autres données.
On note à ce stade que la fermeture explicite du descripteur de fichier à la ligne 15, est en fait redondante pour la simple raison que l'appel à `close()` sera exécuté automatiquement lorsque le descripteur de fichier est rangé lorsque la fonction `main` se termine. Cependant, il est de bonne pratique pour le programmeur novice de faire cet appel explicitement et de savoir ainsi quand s'attendre à ce que la sortie de son programme soit écrite dans un fichier.

L'exécution du code créera un nouveau fichier, `Output.dat`, si ce fichier n'existe pas déjà. Si ce fichier existe, l'exécution du code ci-dessus supprimera le fichier d'origine et écrira un nouveau fichier portant le même nom : le contenu d'origine du fichier sera perdu.
Supposons que, plutôt que de supprimer le fichier s'il existe, nous voulions que notre code ajoute des données à la fin de ce fichier. Ceci serait réalisé en modifiant la ligne 9 du code par :
`std::ofstream write_output("Output.dat", std::ios::app);`

Exemple 1.3

La commande de formatage clé pour les applications de calcul scientifique est la spécification de la précision de la sortie. Ceci est montré dans le code ci-dessous. Le nombre entre parenthèses après les commandes `précision` spécifie le nombre de chiffres significatifs auxquels la sortie est correcte. On note que lorsque la précision est définie sur 10 chiffres significatifs à la ligne 15 du code, seuls huit chiffres significatifs seront imprimés : c'est parce que la variable `x` n'est donnée qu'à huit chiffres significatifs, et donc la précision restante demandée est redondante.

```
1  #include <iostream>
2  #include <fstream>
3
4  int main(int argc, char* argv[])
5  {
6      double x = 1.8364238;
7      std::ofstream write_output("Output.dat");
8
9      write_output.precision(3); // 3 chiffres significatifs
10     write_output << x << "\n";
11
12     write_output.precision(5); // 5
13     write_output << x << "\n";
14
15     write_output.precision(10); // 10
16     write_output << x << "\n";
17     write_output.close();
18
19     return 0;
20 }
```

1.2 Lecture à partir d'un fichier

Lors de la lecture d'un fichier, on doit d'abord déclarer une variable de flux d'entrée d'une manière similaire à la variable de flux de sortie décrite précédemment, puis spécifier le fichier qu'on souhaite lire. Comme pour la sortie vers un fichier, le fichier d'en-tête `fstream` doit être inclus. La lecture du fichier est alors effectuée d'une manière similaire à celle décrite pour la saisie au clavier dans le chapitre précédent, avec `std::cin` remplacé par la variable de flux d'entrée. Le code ci-dessous lit le fichier `Output.dat` créé dans le code précédent. La ligne 9 garantit que

`Output.dat` est sur le disque à l'emplacement correct et avec les privilèges d'accès corrects.

```cpp
#include <cassert>
#include <iostream>
#include <fstream>

int main(int argc, char* argv[])
{
    double x[3], y[3];
    std::ifstream read_file("Output.dat");
    assert(read_file.is_open());
    for (int i=0; i<3; i++)
    {
    read_file >> x[i] >> y[i];
    }
    read_file.close();
    return 0;
}
```

Dans le code ci-dessus, on sait que le fichier à lire contient 3 lignes (6 lignes, si on a exécuté les deux codes d'écriture) et 2 colonnes, et on sait donc lors de l'écriture de ce code que les instructions à l'intérieur de la boucle `for` doivent être exécutées 3 fois. Dans de nombreuses applications de calcul scientifique, on veut lire un fichier, mais on ne connait pas la longueur du fichier à l'avance. Par exemple, on peut savoir qu'un fichier contient une liste des coordonnées d'un nombre inconnu de points en deux dimensions : le fichier a donc deux colonnes, mais un nombre inconnu de lignes. On ne peut pas utiliser de boucle `for` car on ne sait pas combien de fois les instructions de cette boucle doivent être exécutées. Au lieu de cela, on utilise la variable booléenne associée à la variable de flux d'entrée `read_file.eof()`. Cette variable prend la valeur `true` lorsque la fin du fichier est atteinte, et permet, grâce à l'utilisation d'une boucle `while`, de poursuivre la lecture du fichier tant que cette variable prend la valeur `false`. En supposant qu'on sait que le nombre de points est inférieur à 100, on peut y parvenir en utilisant le code suivant (code pas optimal).

```cpp
#include <cassert>
#include <iostream>
#include <fstream>

int main(int argc, char *argv[])
{
    double x[100], y[100];
    std::ifstream read_file("Output.dat");
    assert(read_file.is_open());

    int i = 0;
    while (!read_file.eof())
    {
        read_file >> x[i] >> y[i];
        i++;
    }
    read_file.close();
    return 0;
}
```

1.3 Contrôle du format de sortie

Sortie au format scientifique

Si on utilise le format scientifique, un nombre est écrit comme un produit d'un nombre avec un seul chiffre significatif à gauche de la virgule décimale et une puissance entière de 10, c'est-à-dire que 465.78 au format scientifique est 4.6578×10^2, qui peut être écrit en notation C++ sous la forme `4.6578e2`. Ceci est réalisé par l'utilisation de l'instruction `std::ios::scientific` qui nécessite le fichier d'en-tête fstream.

Toujours afficher un signe + ou –

Le paramètre par défaut pour un flux de sortie est de ne pas imprimer de signe "plus" avant un nombre positif. Pour aligner les nombres dans des colonnes soignées, on peut souhaiter faire précéder un nombre d'un signe "plus"

ou "moins" : ceci est réalisé par l'utilisation de l'instruction `std::ios::showpos` qui nécessite le fichier d'en-tête `fstream`.

Précision au format scientifique de la sortie

Lorsque le format scientifique est utilisé, la précision indiquée est le nombre de chiffres après la virgule, et donc le nombre de chiffres significatifs est supérieur d'un à ce nombre (car il y a un autre chiffre significatif avant la virgule). De plus, lorsque le format scientifique est utilisé, des zéros sont ajoutés après la virgule décimale pour garantir que toutes les sorties ont exactement la même largeur.
Ces techniques de formatage sont illustrées dans le code ci-dessous :

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

int main(int argc, char *argv[])
{
    std::ofstream write_file("OutputFormatted.dat");
    // Ecrire les nombres sous la forme +x.<13 chiffres>e+00 (largeur 20)
    write_file.setf(std::ios::scientific);
    write_file.setf(std::ios::showpos);
    write_file.precision(13);

    double x = 3.4, y = 0.0000855, z = 984.424;
    write_file << x << " " << y << " " << z << "\n";

    write_file.close();
    return 0;
}
```

1.4 Manipulation de fichiers

Exemples :

- Écrire le code le l'exemple 4
- 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 5, 6, 7, 8 et 9.

Exemple 1.4

```cpp
#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 << 2022;

    string texte1 ("Lycée Charles Carnus.");
    fichier << "\t" << texte1;

    fichier << "\n" << 12;

```

```
21      cout << "Fin du programme" << endl;
22
23      return 0;
24  }
```

Exemple 1.5

```
1   #include <fstream>
2   #include <iostream>
3   #include <string>
4   using namespace std;
5
6   int main()
7   {
8       ofstream fichier("C:/Documents/Test.txt", std::ios::app);
9
10      string texte2("BTS SN.");
11      fichier << "\t" << texte2;
12
13      int x = 50;
14      fichier << "\n" << x << "+ 10 = " <<60;
15
16      cout << "Fin du programme" << endl;
17
18      return 0;
19  }
```

Exemple 1.6

```
1   #include <fstream>
2   #include <iostream>
3   #include <string>
4   using namespace std;
5
6   int main()
7   {
8       ifstream fichier("C:/Documents/Test.txt");
9
10      int entier(0);
11      fichier >> entier;
12      cout << "Le nombre entier vaut : " << entier << endl;
13
14      string phrase("");
15      fichier >> phrase;
16      cout << "La phrase est : " << phrase << endl;
17
18      int nombre(0);
19      fichier >> nombre;
20      cout << "Le deuxième nombre : " << nombre << endl;
21
22      return 0;
23  }
```

Exemple 1.7

```
1   #include <fstream>
2   #include <iostream>
3   #include <string>
4   using namespace std;
5
6   int main()
7   {
```

```
    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 1.8

```
#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 1.9

```
#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;
}
```

1.5 Les fichiers binaires

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.
≪ → **f.write()**
≫ → **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 (∀ son type) : `fichier.write((char*)&x, sizeof(x));`
Pour lire une variable y à partir d'un fichier (∀ 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;
```

— **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;
}
```

1.6 Les fichiers $\mathcal{CSV}$

Définition 1.4 : Fichier $\mathcal{CSV}$

— 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 1.10: Création d'un fichier $\mathcal{CSV}$

```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 1.11: $\mathcal{CSV}$ – 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 entieres
    */
    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 1.12: $\mathcal{CSV}$ – Tableau avec 3 colonnes

```cpp
#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 << ",";
        }

        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 1.13: Lecture et duplication d'un fichier $\mathcal{CSV}$

```cpp
#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;
}
```

2. Exercices

Exercice 1

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

— Le nombre le lignes du fichier

— Le nombre de caractères (sans les espaces)

— Le nombre de mots

Exercice 2

- 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

Note 2.1. *Voir l'exemple 9 pour plus de détails.*

Exercice 3

— 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 qui permet d'afficher à l'écran le contenu du fichier **Contact.txt**, en sautant une ligne entre chaque personne.

Exercice 4

— 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 5

Écrire un programme pour créer le fichier CSV représenté dans le tableau suivant :

1	Etudiant1	12	13	14	11
2	Etudiant2	15	10	12	14
3	Etudiant3	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	Etudiant8	3	101	13	12
9	Etudiant9	14	12	10	14
10	Etudiant10	12	12	14	13

Table 1. Données

Exercice 6

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

— Le nombre le lignes du fichier

— Le nombre de caractères (sans les espaces)

— Le nombre de mots