

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

1

# 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 d'être 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: <code>fich</code> )
écriture	<ul style="list-style-type: none"> <li>• tous les types standards : &lt;&lt;</li> <li>exemple: <code>cout &lt;&lt; Variable;</code></li> </ul>	<ul style="list-style-type: none"> <li>• tous les types standards : &lt;&lt;</li> <li>exemple: <code>fich &lt;&lt; Variable;</code></li> <li>• pour les structures : <code>write()</code></li> <li>exemple: <code>Fich.write( char * &amp;structure, sizeof(structure) );</code></li> </ul>
lecture	<ul style="list-style-type: none"> <li>• tous les types standards : &gt;&gt;</li> <li>exemple: <code>cin &gt;&gt; Variable;</code></li> <li>• le type <code>char : get()</code></li> <li>exemple: <code>Variable = cin.get();</code></li> <li>• le type <code>char : _getch() et _getche()</code></li> <li>exemple: <code>Variable = _getche();</code></li> </ul>	<ul style="list-style-type: none"> <li>• tous les types standards : &gt;&gt;</li> <li>exemple: <code>fich &gt;&gt; Variable;</code></li> <li>• le type <code>char : get()</code></li> <li>exemple: <code>Variable = fich.get();</code></li> <li>• pour les structures: <code>read()</code></li> <li>exemple: <code>Fich.read( char * &amp;structure, sizeof(structure) );</code></li> </ul>

Nous allons, dans un premier temps, étudier les entrées-sorties par caractères, 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 **binnaire**.

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=soirie-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

3

# 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, <code>fail</code> est alors vrai et il n'est pas possible d'utiliser <code>fstream</code> |
| <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, <code>fail</code> est alors vrai et il n'est pas possible d'utiliser <code>fstream</code> |
| <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

16

```
#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

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

## Lire un fichier : Un mot à la fois

20

- ▶ Comme pour `cin`, il est possible de lire un mot à la fois
  - ▶ Un mot : Ensemble de caractère séparer 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

The screenshot shows a Microsoft Visual Studio interface. In the top left, there's a terminal window titled "exemple\_4.txt" with the command "mot 42 allo 0.5 56 c". Below it is a larger window titled "Console de débogage Microsoft Visual Studio" showing the output of the program: "mot 42 allo 0.5 56 c". The main area displays the C++ source code for "example\_4.txt".

```
#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

The screenshot shows a Microsoft Visual Studio interface with the source code for "example\_3.txt".

```
#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

## Les fichiers textes

LE FORMAT CSV

26

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(année), 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

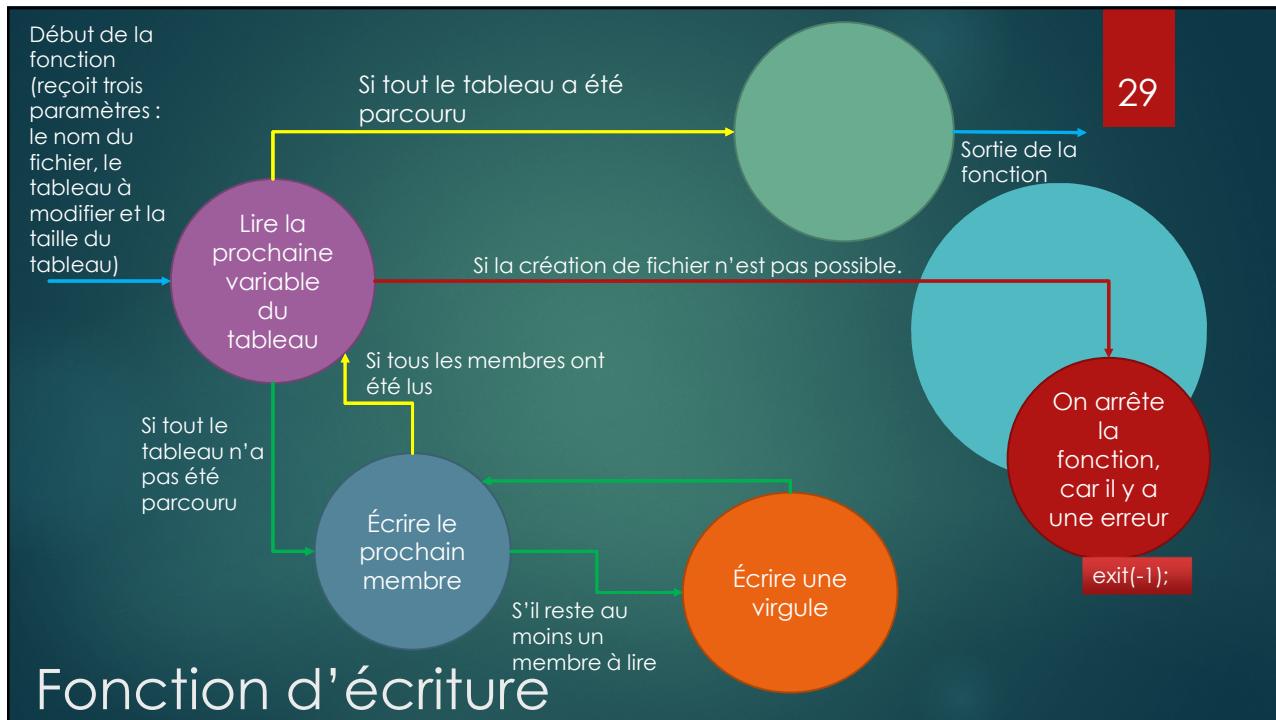
Bidule, Marilou, 14, 02, 1949, 80

Machin, Philippe, 11, 04, 1972, 70

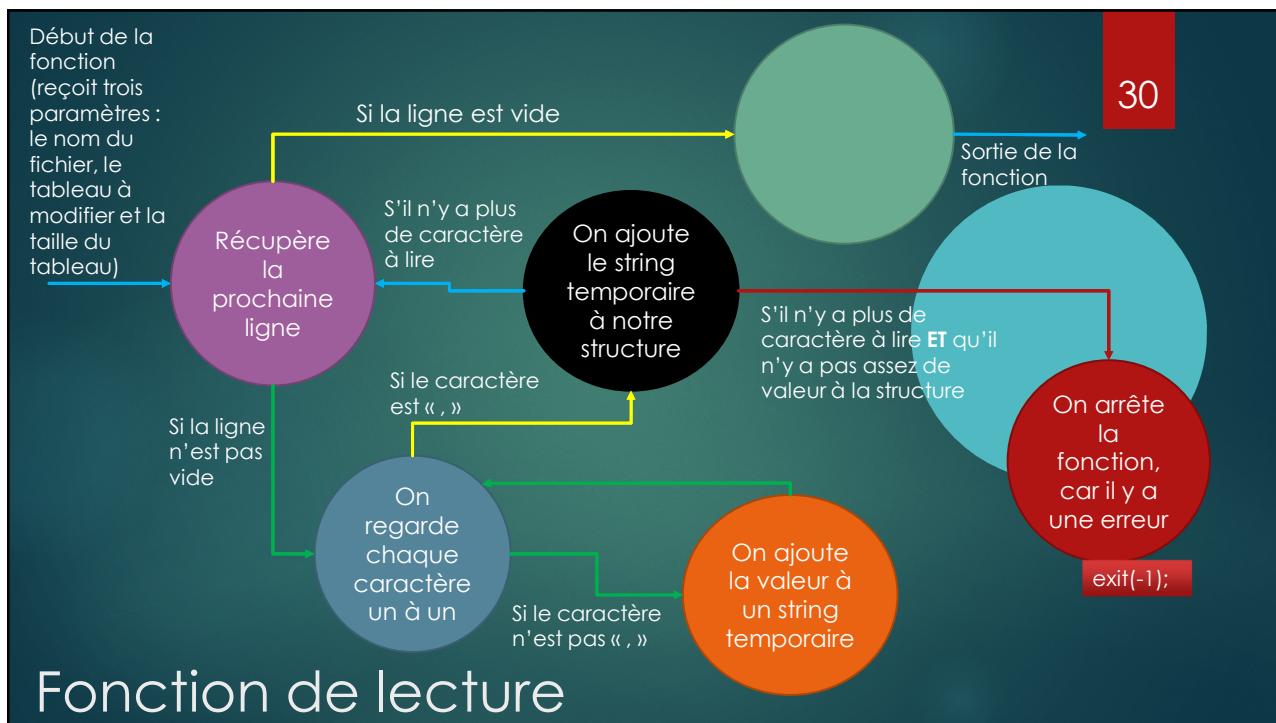
| Personne1 | Contact_s         |
|-----------|-------------------|
| Borduas   | Nom               |
| Godefroy  | Prenom            |
|           | Date de naissance |
| 42        | Téléphone         |

| DateNaissance | Date_s |
|---------------|--------|
| 13            | Jour   |
| 12            | Mois   |
| 1919          | Année  |

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

34

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

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("./fibonacci", 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

## Version tableau

35

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

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("fibo_array.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

## Les fichiers binaires

LIRE UN FICHIER

36

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", ios::in | ios::binary);

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

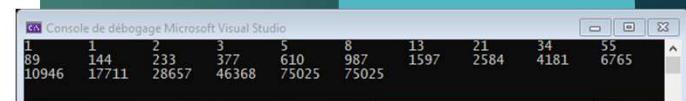
    int nbre_terme = 1, terme;

    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



The screenshot shows the Microsoft Visual Studio Debug Console window titled "Console de débogage Microsoft Visual Studio". It displays the first 10 Fibonacci numbers in a tabular format:

| Index | Value |
|-------|-------|
| 1     | 1     |
| 89    | 144   |
| 10946 | 233   |
| 17711 | 377   |
| 28657 | 610   |
| 46368 | 987   |
| 75025 | 13    |
| 75025 | 21    |
| 1597  | 34    |
| 2584  | 4181  |
| 4181  | 55    |
| 6765  |       |

38

Et on lisait des double ? 39

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

using namespace std;

int main() {
    fstream Fichier;
    Fichier.open("./fibo.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;
    double 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";
}
```

Exception levée  
Run-Time Check Failure #2 - Stack around the variable 'terme' was corrupted.  
Copier les détails | Démarrer la session Live Share...  
Y:\Projet Visual Studio\Fichiers\Debug\BinaireLire.exe  
1 10946 2 28657 5 75025 13 34 89 233 610 1597 4181

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

39

Les fichiers binaires 40

LIRE ET ÉCRIRE DES ENREGISTREMENT

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



47

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas



# CÉGEP DU VIEUX MONTRÉAL

## Programmation multifichiers *partie 2*

Basé sur les notes d'Alain Thiboulot — Hiver 2022

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

### Les variables globales

#### Différence entre variables *locales et globales*

|                               | Globale                                             | Local                    |
|-------------------------------|-----------------------------------------------------|--------------------------|
| Emplacement de la déclaration | Extérieur d'une fonction                            | Intérieur d'une fonction |
| Moment de la création         | Démarrage du programme (avant <code>main()</code> ) | Démarrage de la fonction |
| Visibilité                    | Programme                                           | Fonction                 |
| Durée de vie                  | Programme                                           | Fonction                 |

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

### Pourquoi créer une variable globale ?

- Toujours défini dans un CPP
  - Doit suivre une logique d'organisation
- Contient une information utile à un sous-ensemble du programme
  - Dépasse le simple cadre d'une fonction particulière
- Toujours accessible pour le contenu des définitions du CPP
- Toujours publique, tous les modules y ont accès par défaut
  - Possible de la restringer à un module -> rendre privée
  - Aussi possible de rendre la variable privée à une fonction

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

### Déclaration et définition

- Déclaration = Annonce au compilateur
  - Comme pour les fonctions
  - Annonce que la variable nommée `foo` de type `T` existe quelque part
- Les déclarations vont dans le fichier d'en-tête (`.h`)
- Déclaration de la variable + Définition du type `T` = Vérification des expressions qui l'utilisent
  - Faites lors de la première étape du processus de compilation
- Comme pour les fonctions, la définition doit exister quelque part
  - Pour utiliser un mot, on doit le connaître

## Définition d'une variable

- La définition crée la variable
  - Possible de lui donner une valeur (initialiser)
  - Syntaxe -> Toujours la même qu'en C11
- Exemple :

```
double maVariable = 10;  
Info i = {};  
double notes [5];
```

## Restreindre la variable à un module

- Une variable globale est toujours publique
  - Tout le monde peut la déclarer
  - Tout le monde peut l'appeler
- Le mot clé `static` limite la définition au fichier CPP
- Exemple :

```
static double maVariable;  
static Info i = {};  
static double notes [5] = {};
```

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

- Exemple :

- Est-ce que `reservoir = "15";` fait du sens ?
  - Sans savoir le type de `reservoir` -> impossible
- Est-ce que `Litre reservoir; /* ... */ reservoir = "15";` fait du sens ?
  - Sans savoir la signification de `Litre` -> impossible
- Est-ce que `using Litre = double; /* ... */ Litre reservoir; /* ... */ reservoir = "15";` fait du sens ?
  - Non, car :
    - a. 15 est un string
    - b. `reservoir` est un `Litre`
    - c. `Litre` est un `double`
    - d. Donc 15 n'est pas un `double`

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

### Déclaration d'une variable globale

- La syntaxe d'une déclaration d'une variable globale est plus compliquée que pour les fonctions
- La déclaration se fait toujours avec le mot clé `extern`
  - Impossible d'initialiser la variable -> réservée à la définition
  - Le type doit toujours être disponible à la déclaration -> Pour vérifier
  - La déclaration permet de rendre la variable accessible à tous les modules
  - Toujours placé dans le fichier d'en-tête
- Exemple :

```
extern double maVariable;
extern Info i;
extern double notes [5];
```

### Variable globale... à une fonction

- Toujours possible de limiter la variable globale à une fonction
  - Dans la déclaration locale -> ajouter le mot clé `static`
  - Existe pour toute la vie du programme
  - Accessible uniquement dans la fonction
- Exemple :

```
void foo() {  
    static int compteur = 0;  
    cout << "foo() -> compteur = " << compteur << "\n";  
    // ...  
    compteur++;  
}
```

- Peu fréquent, mais utile pour les gros calculs

## Les constantes globales

- Même chose qu'une variable -> sauf que la valeur ne change pas
- Peut être publique ou privée à un module ou à une fonction
- Les déclarations et les définitions ne changent pas par rapport aux variables globales
- Par défaut, la constante est toujours privées à son modules.

## Déclaration d'une constante globale

- Annonce l'existence de la constante
- La déclaration passe toujours par le mot clé `extern`.
- Jamais initialisé -> réservé à la définition
- Exemple :

```
extern const int max;
extern const double TAUX;
```

## Définition d'une constante globale

- Toujours privé à son module
  - Contient explicitement le mot clé `static`
- Doit toujours avoir une valeur d'initialisation -> sinon mis à 0 ou équivalent
- Exemple :

```
const int MAX = 100;
const double TAUX = 0.15;
```

- Possible de la rendre publique -> doit ajouter le mot clé `extern`
  - Doit être dans le fichier d'en-tête
- Exemple :

```
extern const int MAX = 100;
```

Programmation multifichiers --- Cégep du Vieux Montréal --- Godefroy Borduas

### Mauvais exemple

- Module A :

```
extern const TAILLE = 5;
```

- Module B :

```
int Tableau[TAILLE];
```

- Impossible -> Car TAILLE n'est pas connue à la compilation

Programmation multifichiers --- Cégep du Vieux Montréal --- Godefroy Borduas

### Bon exemple

- Module A :

```
const TAILLE = 5;  
int Tableau[TAILLE];
```

- Possible -> Car TAILLE est connue à la compilation

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

- Possible de limiter à une fonction -> utiliser le mot clé `static`
- Exemple :

```
void foo()
{
    static const int BAR = 42;
    // ...
}
```

### Équivalence entre définition sans `static` et avec

- Les deux définitions suivantes sont strictement équivalentes

```
const int MAX = 100;
static const int MAX = 100;
```

Programmation multifichiers — Cégep du Vieux Montréal — Godefroy Borduas

### Cas particulier avec les tableaux

- La taille d'un tableau doit toujours être constante
  - Sinon, allocation dynamique
- Doit être connue à la compilation
- Peut être réalisé par une constante mais la constante doit être connue à la création (définition) du tableau
  - Donc -> doit être dans le même module
  - Chaque module est compilé indépendamment

## Les différents types d'erreur

- Compilation -> Deux grande étapes **successives**
  - i. Précompilation, Vérification, Génération du code (obj)
  - ii. Vérification et Edition des liens (entre les fichiers obj), Assemblage final du programme.exe
- L'étape 2 ne peut être accomplie sans le succès de la première
- Chaque étape entraîne des erreurs

## Compilation et erreur

- Fenêtre de sortie de VS

```
1>----- Début de la génération : Projet : Travaux, Configuration : Debug x64 -----
1>main.cpp
1>sondage.cpp
1>statistic_01.cpp
1>statistic_02.cpp
1>statistic_03.cpp
1>e:\code\statistic_03.cpp(13): error C2065: 'reponses' : identificateur non déclaré
1>statistic_04.cpp
1>statistic_05.cpp
1>statistic_06.cpp
1>statistic_07.cpp
1>statistic_08.cpp
1>e:\code\statistic_08.cpp(9): error C2065: 'nbPersonne' : identificateur non déclaré
1>statistic_09.cpp
1>Génération de code en cours...
1>Génération du projet "Travaux.vcxproj" terminée -- ÉCHEC.
===== Génération : 0 a réussi, 1 a échoué, 0 mis à jour, 0 a été ignoré =====
```

### Liste des erreurs

- Ligne 13 du fichier `statistic_03.cpp`
- Ligne 9 du fichier `statistic_08.cpp`

### Une fois corrigé

```
1>----- Début de la génération : Projet : Travaux, Configuration : Debug x64 -----
1>statistic_03.cpp
1>statistic_08.cpp
1>Génération de code en cours...
1>Travaux.vcxproj -> C:\Users\Alain\source\repos\Travaux\x64\Debug\Travaux.exe
===== Génération : 1 a réussi, 0 a échoué, 0 mis à jour, 0 a été ignoré =====
```

- La compilation est incrémentale

### Quelle est cette erreur ?

- Fenêtre de sortie de VS

```
1>----- Début de la régénération globale : Projet : Travaux, Configuration : ebug x64 -----
1>main.cpp
1>sondage.cpp
1>statistic_01.cpp
1>statistic_02.cpp
1>statistic_03.cpp
1>statistic_04.cpp
1>statistic_05.cpp
1>statistic_06.cpp
1>statistic_07.cpp
1>statistic_08.cpp
1>statistic_09.cpp
1>Génération de code en cours...
1>main.obj : error LNK2019: symbole externe non résolu "int __cdecl stat2(bool * const)" (?stat2@@YAHQEAN@Z) référencé dans la fonction main
1>statistic_05.obj : error LNK2001: symbole externe non résolu "int BIDON" (?BIDON@@3HA)
1>C:\Users\Alain\source\repos\Travaux\x64\Debug\Travaux.exe : fatal
error LNK1120: 2 externes non résolus
1>Génération du projet "Travaux.vcxproj" terminée -- ÉCHEC.
===== Régénération globale : 0 a réussi, 1 a échoué, 0 a été ignoré =====
```

## Deux erreurs de liens -> Étape 2 de la compilation

- Toutes les erreurs liens commencent par le code LNKxxxx
  - Pour plus de détail sur le type d'erreur, cliquez sur le code
- 1. Dans main.obj -> Erreur LNK2019 -> Sur int stat2(bool \* const) -> Dans la fonction main
  - La définition de la fonction stat2 n'existe pas dans le contexte
    - Soit la fonction n'est pas définie
    - Soit le prototype est différent
- 2. Dans statistic\_05.obj -> Erreur LNK2001 -> Sur int BIDON -> Quelque part dans le module
  - Aucune variable BIDON de type int existe
    - Soit que la déclaration et la définition est différente
    - Soit que la variable est privée

## Quelle est cette erreur ?

- Fenêtre de sortie de VS

```
1>----- Début de la régénération globale : Projet : Travaux, Configuration : ebug x64 -----
1>main.cpp
1>sondage.cpp
1>statistic_01.cpp
1>statistic_02.cpp
1>statistic_03.cpp
1>statistic_04.cpp
1>statistic_05.cpp
1>statistic_06.cpp
1>statistic_07.cpp
1>statistic_08.cpp
1>statistic_09.cpp
1>Génération de code en cours...
1>statistic_04.obj : error LNK2005: "int __cdecl stat2(bool * const)" (?stat2@@YAHQEAN@Z) déjà défini(e) dans statistic_02.obj
1>C:\Users\Alain\source\repos\Travaux\x64\Debug\Travaux.exe : fatal error
NK1169: un ou plusieurs symboles définis à différentes reprises ont été rencontrés
1>Génération du projet "Travaux.vcxproj" terminée -- ÉCHEC.
===== Régénération globale : 0 a réussi, 1 a échoué, 0 a été ignoré =====
```

### Une erreur de lien -> Étape 2 de la compilation

1. Dans `statistic_04.obj` -> Erreur LNK2005 -> Sur `int stat2(bool * const)` ->  
Défini dans `statistic_02.obj`
  - Une fonction ne peut jamais être définie deux fois
    - Laquelle faut-il appeler ?
  - Même problème pour les définition multiples des variables et des constantes

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

# Programmation multifichiers

Basé sur les notes d'Alain Thiboutot – Hiver 2022



**CÉGEP DU  
VIEUX MONTRÉAL**

1

1

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Principe de base d'un compilateur

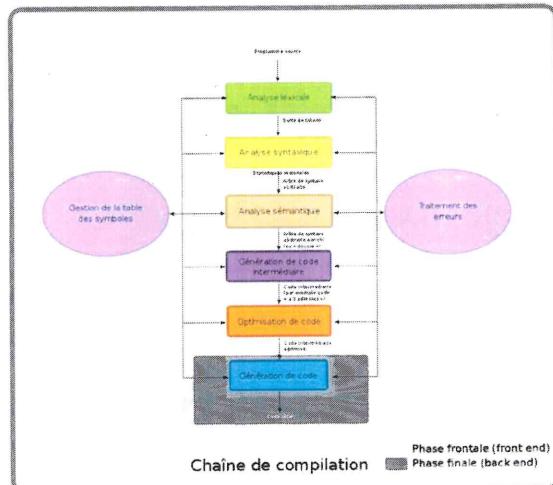
- Traduire un langage source vers un langage cible
- Objectifs :
  - Faciliter l'écriture des instructions
  - Optimiser les opérations
  - Modifier les paradigmes d'écriture
  - Etc.

2

2

1

## Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas



- C++ : Conversion vers Assembleur (code machine)

3

3

## Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Processus de compilation C++

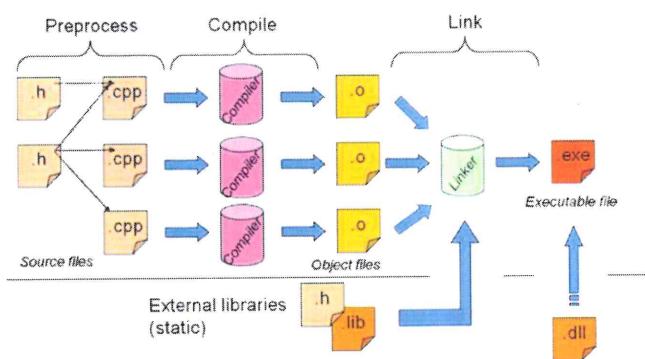


FIGURE 1.1 – Création d'un exécutable lors de la compilation en C++

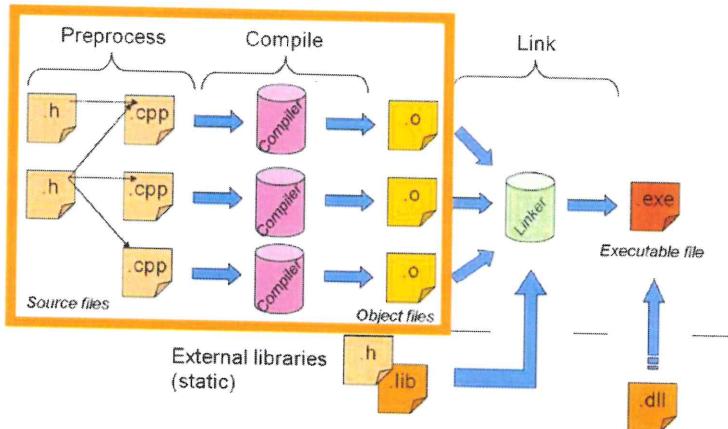
4

4

2

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## 1. Précompilation, vérification et génération de code machine



5

5

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

### A. Étape de précompilation -> Préparation des fichiers CPP selon les directives

- Lecture de `nom.cpp`
- Exécution des commandes de précompilation (`#pragma`, `#include`, `#define`, etc.)
- Écriture d'un fichier transformé en fichier `nom.cpp+`
  - Le fichier n'est pas visible. Il est **interne** au compilateur.
- Exemple : l'instruction `#include <fichier>` est remplacé par le contenu du fichier
- Exemple avec le français : Rassemblez tous les paragraphes ensemble et appliquez les styles

6

6

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## B. Étape de vérification -> Conformité du texte avec les règles

- Lecture du fichier `main.cpp+`
  - Maintenant conforme aux traitements de préprocessus
- Exécution des analyseurs :
  - Lexicale : Vérification des mots -> Existance dans le langage ?
  - Syntaxique : Vérification des phrases -> Conforme au langage ?
  - Sémantique : Vérification du sens -> Veut dire quelque chose ?
- Au besoin, renvoie les erreurs rencontrées
- Exemple avec le français : Corriger son texte

7

7

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## C. Étape de génération du code machine

- Traduire le C++ en Assembleur
- Lecture de l'analyse du fichier `main.cpp+`
- Écriture du fichier `nom.o`
- Répertorier toutes les définitions du programme
  - Constante, Variables, fonctions, etc.
  - Utilisé durant l'édition de lien (étape 2)
- Exemple avec le français : Mise au propre et publication du texte

8

8

## Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

```
#include <stdio.h>
int main()
{
    printf("hello world");
    return 0;
}
```

Code C/C++

```
.file "test.c"
.section .rodata
.LC0:
.string "hello world"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushq %rbp
.cfi_offset %rbp, -16
.cfi_offset $, -16
movq %rsp, %rbp
.cfi_offset %rbp, 16
.cfi_offset %rbp, -16
movl $.LC0, %edi
movl $.S, %eax
call printf
movl $9, %eax
popq %rbp
.cfi_offset %rbp, 8
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
.section .note.GNU-stack,"@progbits"
```

Code Assembleur

9

9

## Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

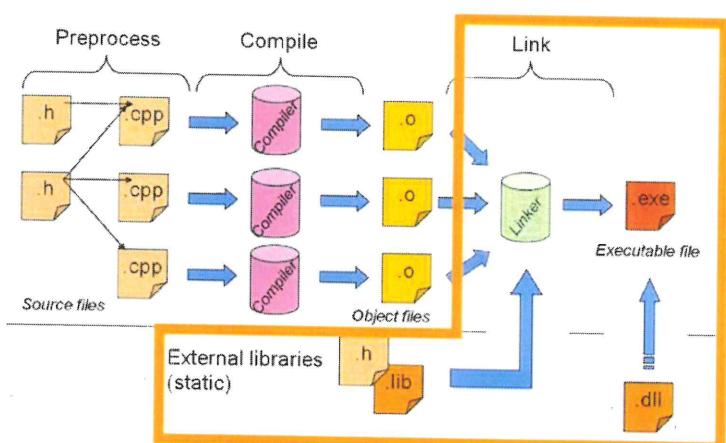
**2. Édition de lien et création de l'exécutable**

FIGURE 1.1 – Création d'un exécutable lors de la compilation en C++

10

10

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## A. Vérification des définitions et édition des liens

- Si tous les fichiers `.o` ont été générés
- Liaison des références à une entité (constante, variable, fonction) à une définition
  - Une définition ne peut pas être multiple -> Pas possible d'avoir deux liens
- Rapporte les erreurs de liaison

## B. Assemblage des fichiers `.o`

- Si toutes les références sont liées, écriture d'un fichier exécutable

11

11

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Project 1

- Fichiers du projet
  - [main.cpp](#)
  - [stats.cpp](#)
  - [stats.h](#)
- Les trois fichiers doivent présents **dans le projet VS**
- [Projet Visual Studio](#)

12

12

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Pourquoi diviser le projet ?

- Faciliter
  - la lecture
  - le développement
  - la mise à jour
  - la réutilisation (ex. `cvm 21.h` et `cvm 21.cpp`)
- Même idée que les fonctions, mais du côté organisation
- Quel arrangement est le meilleur pour trouver les listes ?
  1. `syntaxe.cpp`, `list.cpp`, `net.cpp`, `main.cpp`
  2. `programme.cpp`

13

13

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

- Aucune limite au nombre et à la taille des fichiers
  - Importance : Être logique et cohérent
- Par convention, le programme contient toujours le fichier `main.cpp`
  - Porte d'entrée au programme
  - Contient uniquement la fonction `int main()`

14

14

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Visibilité des fonctions

- **Visibilité**: Principe d'accès à une entité selon son module. Il existe deux types de visibilité en C++ :
  - **Publique** -> Vue et accessible par tous
  - **Privée** -> Vue et accessible localement à son module (~ son fichier)
- Par défaut, une fonction est **publique**
  - Tous les modules peuvent l'appeler
  - Peut-être changer par le mot clé **static**

15

15

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Toujours 2 fichiers en C++

- Programmation multifichier <=> Division en module
  - Chaque module fournit un service précis (ex. liste, affichage, etc.)
- En C++, tous les modules sont en **pair** -> fichier **.CPP** et **.h**
  - Fichier **source** (**.cpp**) -> Définition des fonctions, des variables et des constantes
  - Fichier **d'en-tête** (**.h**) -> Déclaration des fonctions, des variables et des constantes
- Le nom doit toujours être **significatif**

16

16

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Règles à respecter

1. Les définitions vont **TOUJOURS** dans le fichier **source**
2. Les déclarations vont **TOUJOURS** dans le fichier **d'en-tête**
  - Possible d'ajouter du C++ dans le fichier d'en-tête, mais dois respecter le principe précédent

17

17

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Le fichier source **.CPP**

- Toujours constitué des corps (définition) des fonctions
- Toujours liés à un fichier **d'en-tête**
  - Sauf pour **main.cpp**
- Contiens la liste des inclusions (déclaration à utiliser), Ordre :
  1. Fichier de la bibliothèque standard
  2. Fichier d'en-tête du projet
  3. Fichier d'en-tête lié au fichier source
- Ajouter **uniquement** les fichiers **utilisés** dans les déclarations

18

18

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

```
#include <iostream>      // inclure d'abord les fichiers d'en-tête
// des librairies standards absolument essentielles au cpp
#include ... // autres include standards

#include "cvm 21.h"
// inclure ensuite les fichiers d'en-tête du projet absolument essentiels au cpp
#include ... // autres include du projet

#include "menu.h" // inclure ensuite son propre fichier d'en-tête

... // et les définitions des différents éléments du code:
// types, const, variables, fonctions
```

19

19

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Le fichier d'en-tête (.h)

- Toujours inclure la ligne suivante en **premier**
  - **#pragma once**
  - Expliqué un peu plus tard
- Contiens les déclarations
  - Contiens uniquement les déclarations de fonctions publiques
- Attention, l'écriture des déclarations est une source d'erreur
  - Le compilateur les utilise pour définir le lien entre les fichiers
  - Logique ! Pour utiliser quelque chose, il faut qu'il soit décrit

20

20

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Précompilation et directive `#include`

- La directive `#include` remplace la ligne par le contenu du fichier mentionné
- S'applique toujours avant la vérification
- Exemple : Fichier `stat.h`

```
#pragma once

double moyenne(double notes[], size_t size);
double variance(double notes[], size_t size);
double ecart_type(double notes[], size_t size);
```

21

21

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

- Exemple : Fichier `stat.cpp` (avant l'application de `#include`)

```
#include <math.h>
#include "stats.h"

// ...
double moyenne(double notes[], size_t size)
{
    return size ? somme(notes, size) / size : 0; // test de division pas zéro
}

// ...
```

22

22

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

- Exemple : Fichier `stat.cpp` (après l'application de `#include`)

```
#include <math.h>
// INCLUDE STATS_H
#pragma once

double moyenne(double notes[], size_t size);
double variance(double notes[], size_t size);
double ecart_type(double notes[], size_t size);
// END INCLUDE STATS_H

// ...
double moyenne(double notes[], size_t size)
{
    return size ? somme(notes, size) / size : 0; // test de division pas zéro
}
// ...
```

23

23

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Pourquoi les fichiers d'en-tête ?

- Pour utiliser la fonction `f()`, le compilateur doit la connaître **dans le fichier qui l'appelle**
  - Chaque fichier source est compilé **séparément**
- Le fichier d'en-tête fournit cette information
- Tous les fichiers sources qui utilisent une fonction du module **A** doivent inclure le fichier d'en-tête de ce module.

24

24

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Emplacement dans l'explorateur de fichier

- Placer dans le répertoire de projet
- Si le répertoire est divisé, les fichiers sources et d'en-têtes ne sont jamais séparés

## Projet Visual Studio

- Chaque fichier est placé dans le filtre approprié
- Facilite la recherche
- On réfère les fichiers d'en-tête toujours dans un chemin relatif.

25

25

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Project 2

- Tous les fichiers doivent présents **dans le projet VS**
- [Projet Visual Studio](#)

26

26

## Le fichier d'en-tête et les structures

- Comme avant, les déclarations (prototype) de fonction sont dans le fichier `.h`
- La déclaration de structure s'y trouve aussi
  - Placer avant les déclarations de fonction
  - Peut-être placé dans un fichier séparé
  - Toutes les fonctions qui s'y réfèrent doivent inclure ce fichier d'en-tête

27

27

## Bon exemple 1

- Fichier `info_1.h`

```
struct Info {  
    ...  
}  
  
void afficher(Info i);
```

28

28

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Bon exemple 2

- Fichier `info_struct.h`

```
struct Info {
    ...
}
```

- Fichier `info_2.h`

```
#include "info_struct.h"
void afficher(Info i);
```

29

29

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Les structures, mais où ?

1. Utile dans une fonction ?
  - Défini dans la fonction en question
2. Utile dans plusieurs fonctions ou dans un ou des prototypes **d'un module** ?
  - Si publice -> Dans le fichier `.h` du module
  - Si privé -> Dans le fichier `.cpp` du module
3. Utile dans **plusieurs modules** ?
  - Dans un fichier `structure.h` (il n'y aura pas de `.cpp` à ce moment)

30

30

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## La directive `#pragma once`

- Remarque -> Une définition ne doit pas être incluse deux fois
- La directive `#pragma once` indique au compilateur de ne pas ajouter deux fois le contenu
  - Dois toujours être placé à la première ligne
- Tous les fichiers d'en-tête de la bibliothèque standard l'implémentent
  - Pour vérifier :
    1. Faite clique droit sur le nom du fichier
    2. Accéder au fichier

31

31

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Utile, mais pas standard

### ATTENTION ! Contenu bonus

- `#pragma once` n'est pas standard au langage
  - Certains compilateurs ne l'utilisent pas
- Méthode standard : Entourer le contenu d'une condition

```
#ifndef CONSTANTE_UNIQUE
#define CONSTANTE_UNIQUE

// Déclaration et compagnie

#endif
```

32

32

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Project 3

- Tous les fichiers doivent présents **dans le projet VS**
- [Projet Visual Studio](#)

33

33

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Objectif

- Créer une description de voiture et de bateau
- Besoins de plusieurs définitions :

## Liste de combustible

- Énumération de choix :
  - Essence
  - Diesel
  - Hydrogène
  - Éthanol

34

34

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Quelques redéfinitions de nom

- Redéfinition de double :
  - **Litre et Chevaux-Vapeur**
  - **Kilomètre et Kilomètre-heure**
  - **Litres aux 100 km et litre par heure**
  - **Heures et Noeuds**
- Syntaxe à utiliser :

```
using <redéfinition> = <type>;
```

35

35

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Moteur

| Nom       | Type          | Description               |
|-----------|---------------|---------------------------|
| carburant | Combustible   | Type de carburant utilisé |
| puissance | Cheval-Vapeur | Puissance fournie         |
| marque    | String        | Nom de la marque          |

36

36

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Voiture

| Nom           | Type            | Description               |
|---------------|-----------------|---------------------------|
| moteur        | Moteur          | Moteur de la voiture      |
| reservoir_max | Litre           | Taille max du réservoir   |
| conso_ville   | LitreAu100KM    | Consommation en ville     |
| conso_route   | LitreAu100KM    | Consommation sur la route |
| vitesse_max   | Kilomètre-heure | Vitesse maximale          |
| réservoir     | Litre           | Taille du réservoir       |

37

37

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Bateau

| Nom             | Type           | Description                            |
|-----------------|----------------|----------------------------------------|
| moteur          | Moteur         | Moteur de la voiture                   |
| reservoir_max   | Litre          | Taille max du réservoir                |
| conso_plein_gaz | LitresParHeure | Consommation avec les moteurs à fond   |
| conso_croisière | LitresParHeure | Consommation à la vitesse de croisière |
| vitesse_max     | Noeuds         | Vitesse maximale                       |
| réservoir       | Litre          | Taille du réservoir                    |

38

38

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Fonctions à ajouter

### Voiture

- Avoir l'autonomie, en Kilomètre, en ville
- Avoir l'autonomie, en Kilomètre, sur la route
- Avoir la consommation d'essence, en litre au 100 km, sur une distance parcourue

39

39

Programmation multifichiers – Cégep du Vieux Montréal – Godefroy Borduas

## Bateau

- Avoir l'autonomie, en heure, en vitesse de croisière
- Avoir l'autonomie, en heure, à pleine vitesse (*plein gaz*)
- Avoir la consommation d'essence, en litre par heure, pour une durée de parcours

40

40

# LES LISTES CHAÎNÉES

Hiver 2023  
Godefroy Borduas

1

## OBJECTIF

Trouver une façon de conserver les objets selon différents critères et de les ajouter ou de les supprimer facilement.

2



## PLAN DE TRAVAIL

3

J'achète **4 crayons**. Comment les range-t-on ?  
Construisons une boîte avec une case pour chaque crayon.

Mais si j'achète un **nouveau** crayon, comment puis-je le ranger ?

Construisons une nouvelle case.  
Comment ajoute-t-on la nouvelle case aux autres ?

Nous pouvons les lier par une corde.

Construire un modèle visuel

Voir d'autres exemples

Expliquer les inventaires dans les RPG

Programmation des listes chaînées

4

Illustration de la liste chaînée



« La liste chaînée est une **structure de donnée** qui permet de conserver une **collection ordonnée** d'objets du **même type** de taille **non définie**. »

## LA LISTE CHAÎNÉE



5

- Une liste de client, trié par ordre alphabétique
- Les mots du dictionnaire
- L'inscription à un cours
- Un tableau de pointage de tous les joueurs
- L'inventaire du joueur dans un RPG.

## QUELLES SONT LES UTILITÉS DE LA LISTE CHAÎNÉE ?



6

Avec une liste chaînée, on peut ajouter autant d'items que l'on veut.

On peut ajouter des critères de tri ou de recherche.

On peut ajouter des conditions à l'ajout. Est-ce que le poids maximal est atteint ?

D'autres exemples ?

## COMMENT PROGRAMME-T-ON UN INVENTAIRE DE JOUEUR ?



7

## PROGRAMMONS L'EXEMPLE DES CRAYONS

Première étape : Le crayon



8

- ▶ Il est décrit par...
  - ▶ Sa couleur -> String
  - ▶ Le diamètre de sa mine (7 mm ou 5 mm) -> int
  - ▶ La force de sa mine (HB) -> String

```
#include <iostream>
#include <conio.h>

using std::cout; using std::string;

struct Crayon
{
    string Couleur;
    int Diametre;
    string Force;
};
```

## QU'EST-CE QU'UN CRAYON ?



9

## PROGRAMMONS L'EXEMPLE DES CRAYONS

Deuxième étape : La liste



10

► D'une « boîte » contenant le Crayon  
 ► D'un lien vers la « boîte » suivante  
 ► D'un indicateur de sa taille  
 ► D'une fonction d'ajout  
 ► D'une fonction de retrait  
 ► D'une fonction pour récupérer  
 ► D'une fonction pour vider la liste

**Où est-ce qu'on retrouve la première « boîte » ?**  
 Dans une variable *Entête*

**Qu'est-ce qu'on entend par « boîte » ?**  
 Un objet qui contient quoi ?

## DE QUOI EST CONSTITUÉE UNE LISTE ?

Construire un modèle visuel      Voir d'autres exemples      Expliquer les inventaires dans les RPG      Programmation des listes chaînées

11

► Le Crayon  
 ► Le lien vers la « boîte » suivante  
 ► Une méthode pour se rendre à la « boîte » suivante.  
 ► Une méthode pour récupérer le crayon.

**Qui doit avoir accès à la « boîte » ?**  
 Seulement la liste...  
 La « boîte » existe seulement dans la liste.

**Qu'est-ce qu'on programme en premier ?**  
 La « boîte », car elle est utilisée par la liste.

**Est-ce qu'il a des fonctions pour la « boîte » qui sont utiles ?**  
 Non, si la boîte n'existe que pour la liste.

## QU'EST-CE QUI CONSTITUE LA FAMEUSE BOÎTE ?

Construire un modèle visuel      Voir d'autres exemples      Expliquer les inventaires dans les RPG      Programmation des listes chaînées

12

```
struct Noeud
{
    Crayon valeur;
    Noeud* suivant;
};
```

**UN NOEUD OBJET**

Construire un modèle visuel
Voir d'autres exemples
Expliquer les inventaires dans les RPG
Programmation des listes chaînées

13

- ▶ Structure contenant la référence vers le premier noeud
- ▶ Contient aussi le nombre d'éléments de la liste

```
struct Liste
{
    Noeud* racine;
    size_t taille;
};
```

**LA LISTE : L'EN-TÊTE ET LA TAILLE DE LA LISTE**

Construire un modèle visuel
Voir d'autres exemples
Expliquer les inventaires dans les RPG
Programmation des listes chaînées

14

- ▶ Demander la mémoire
- ▶ Établir la taille à zéro
- ▶ Établir le nœud racine à NULL
- ▶ Retourner l'adresse du début de la liste

```
Liste** CreerListe() {
    Liste* liste = new Liste;
    liste->racine = NULL;
    liste->taille = 0;

    return &liste;
}

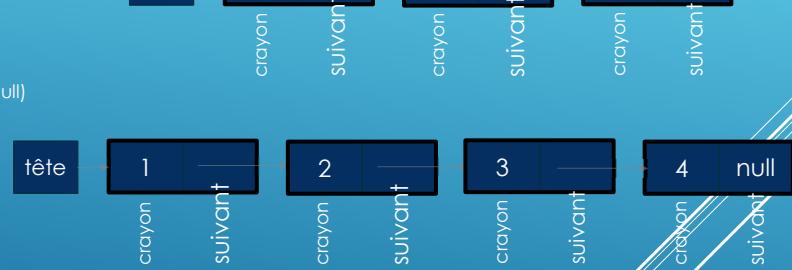
int main() {
    Liste** liste = CreerListe();
}
```

## LA LISTE : CRÉER LA LISTE



15

- ▶ Que doit-on faire ?
- ▶ Créer un nœud pour le nouveau crayon
- ▶ L'ajouter à la fin
  - ▶ Sera lié au dernier nœud
  - ▶ Quel est le dernier nœud ?
  - ▶ Celui sans lien (suivant == null)
- ▶ Incrémenter de 1 la taille



## LA LISTE : AJOUT D'UN CRAYON (1/3)



16

## LA LISTE : AJOUT D'UN CRAYON (2/3)

Complexité temporelle :  
 $O(n)$

```
Liste** Ajouter(Liste** liste, Crayon c) {
    Noeud* noeud = new Noeud;
    noeud->valeur = c;
    noeud->suivant = NULL;

    // Cas où la liste est vide
    if ((*liste)->racine == NULL) {
        (*liste)->racine = noeud;
    } else {
        Noeud * node_liste = (*liste)->racine;
        while (node_liste->suivant != NULL)
            node_liste = node_liste->suivant;

        node_liste->suivant = noeud;
    }

    (*liste)->taille++;
}

return liste;
```

Construire un modèle visuel

Voir d'autres exemples

Expliquer les inventaires dans les RPG

Programmation des listes chaînées

17

## LA LISTE : AJOUT D'UN CRAYON (3/3)

```
int main() {
    Liste** liste = CreerListe();
    Ajouter(liste, { "Noir", 7, "HB" });
}
```

Construire un modèle visuel

Voir d'autres exemples

Expliquer les inventaires dans les RPG

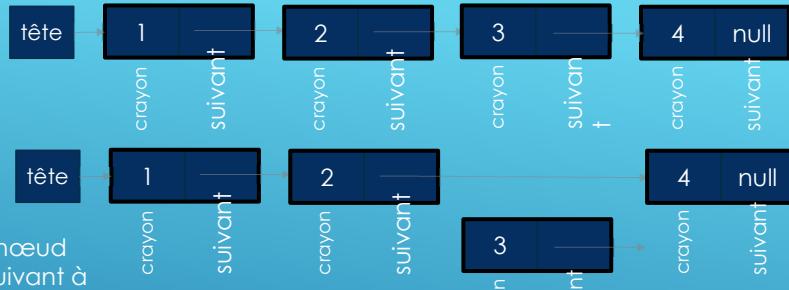
Programmation des listes chaînées

18

## LA LISTE : RETRAIT D'UN CRAYON (1/2)

► Que doit-on faire ?

- ▶ Changer le lien suivant du noeud précédent pour le noeud suivant à celui à supprimer
- ▶ On se fit à la position dans la liste.
  - ▶ Plus simple à chercher (boucle for)
- ▶ Désallouer la mémoire du noeud
- ▶ Décrémenter de 1 la taille



Construire un modèle visuel

Voir d'autres exemples

Expliquer les inventaires dans les RPG

Programmation des listes chaînées

19

```
Liste** Retirer(Liste** liste, int position) {
    if (position >= (*liste)->taille)
        return liste;

    Noeud* node = (*liste)->racine;
    for (size_t i = 0; i < position - 1; i++)
    {
        node = node->suivant;
        if (node == NULL && i < position)
            return liste;
    }

    Noeud* naretirer = node->suivant;
    node->suivant = node->suivant->suivant;
    delete naretirer;
    (*liste)->taille--;
    return liste;
}
```

## LA LISTE : RETRAIT D'UN CRAYON (2/2)

**Complexité temporelle :**  
En pire cas  $O(n)$

```
int main() {
    Liste** liste = CreerListe();
    Ajouter(liste, { "Noir", 7, "HB" });
    Ajouter(liste, { "Bleu", 7, "HB" });

    Retirer(liste, 1);
}
```

Construire un modèle visuel

Voir d'autres exemples

Expliquer les inventaires dans les RPG

Programmation des listes chaînées

20

- ▶ Que doit-on faire ?
  - ▶ Parcourir la liste jusqu'à la position
  - ▶ Retourner le crayon s'il existe
  - ▶ Retourner NULL sinon

## LA LISTE : RÉCUPÉRER UN CRAYON (1/2)



21

## LA LISTE : RÉCUPÉRER UN CRAYON (2/2)

```
Crayon* Rechercher(Liste **liste, int position)
{
    if (position >= (*liste)->taille)
        return NULL;

    Noeud* node = (*liste)->racine;
    for (size_t i = 0; i < position; i++)
    {
        node = node->suivant;
        if (node == NULL && i < position)
            return NULL;
    }

    return &node->valeur;
}
```

```
int main()
{
    Crayon* exemple = Rechercher(&liste, 0);
    if (exemple != NULL) {
        cout << exemple->Couleur
            << ", " << exemple->Diametre
            << " mm, " << exemple->Force
            << "\n";
        exemple->Couleur = "Bleu";
    }

    Crayon* ex2 = Rechercher(&liste, 0);
    if (ex2 != NULL)
        cout << ex2->Couleur << ", "
            << ex2->Diametre << " mm, "
            << ex2->Force << "\n";
    /**
}
```



22

- ▶ Il est toujours nécessaire de libérer la mémoire allouée
- ▶ Le programme alloue deux mémoires
  - ▶ La liste
  - ▶ Les nœuds
- ▶ Comment faire
  - ▶ Parcourir la liste (fonction récursive)
  - ▶ Libérer du dernier nœud au premier
  - ▶ Libérer la liste

## LA LISTE : LIBÉRER UNE LISTE (1/2)



23

```
static void LibererNoeud(Noeud* noeud) {
    if (noeud->suivant != NULL)
        LibererNoeud(noeud->suivant);

    delete noeud;
}

void Liberer(Liste** liste) {
    LibererNoeud((*liste)->racine);
    delete (*liste);
}
```

**Complexité temporelle :**  
 $O(n)$

```
int main() {
    Liste** liste = CreerListe();
    /*
     ...
    */
    Liberer(liste);
}
```

## LA LISTE : LIBÉRER UNE LISTE (2/2)



24

## AVANTAGE ET DÉSAVANTAGE DE LA LISTE CHAÎNÉE

25

### Avantage

- La taille est flexible
- Possibilité de suppression d'élément
- Possibilité de réorganiser la liste (pas vu dans les diapos)

### Inconvénients

- L'opération d'ajout est longue
- L'opération de retrait est longue
- L'opération de recherche est longue
- Récupérer un item est long

Est-il  
possible  
d'améliorer  
le temps  
d'ajout et  
de retrait ?

26

# LA CRÉATION DE SES PROPRES TYPES LES STRUCTURES

Module 2

420-C21-IN Programmation II

Godefroy Borduas – Automne 2021

1

## PLAN DE LEÇON

- Qu'est-ce qu'une structure ?
- Comment définir une structure ?
- Existe-t-il une convention de codage ?
- Comment utiliser une structure ?
- Utiliser cin ou cout sur les structures
- L'impact sur le dictionnaire de donnée
- Les structures dans des structures (Inception)
- Utiliser des tableaux de structure
- Exercices

2

# QU'EST-CE QU'UNE STRUCTURE ?

3

## ÇA TOUCHE LA MÉMOIRE

- Un des problèmes importants est l'organisation de l'information
- En C11, vous avez vu les types « simples » et les types « agrégats »
  - Type simple : Type de variable de base (nombre, caractères)
  - Type agrégats : Type réunissant plusieurs valeurs de type simple (tableau, string)
- Très utile pour des problèmes simples, exemple :
  - Calcul sur des entiers (trois entiers)
  - Collecter les notes d'une classe (tableau de notes)

4

2

## MAINTENANT, PRENONS UNE INFORMATION PLUS COMPLEXE

- Le profil d'une personne étudiante :
  - Nom
  - Prénom
  - Matricule
  - Téléphone
  - Nombre de cours contributoire
  - Moyenne générale
- Maintenant, créez un programme informatique afin de gérer le profil pour une personne étudiante
  - Modifier le téléphone
  - Modifier la moyenne générale

5

QU'AVEZ-VOUS REMARQUÉ ?

6

## ALLONS-Y AVEC UNE EXPÉRIENCE DE PENSÉE

- Imaginez que le système doit gérer, en tout temps, dix personnes étudiantes
- Comment allez-vous modifier le code ?
- Qu'est-ce que vous en concluez ?
- Intuitivement, y a-t-il une solution ?

7

## PENSONS EN TERMES D'ENTITÉ

- Et si la personne étudiante était une entité informatique
  - Entité : Élément informatique manipulable et traitable (ex. int, float, etc.)
- Dans ce cas, la personne étudiante aura des **sous-entités**
- En langage informatique, on parle de **structure**

8

## DÉFINITION

Une **structure** est un type informatique composé ou dérivé des types simples. Ce type permet de manipuler un ensemble d'information comme un seul objet.

Similaire à un tableau, la structure réunit au même endroit toutes les informations relatives à une même entité.

Une fois créée, la structure devient un type C++ comme les int, les floats et compagnie.

9

## À RESSEMBLE UNE STRUCTURE

- Revenons à la personne étudiante, il est possible de créer une structure pour réunir toutes les informations au même endroit.
- La structure ressemble à ceci  
Personne étudiante
  - Nom (string)
  - Prénom (string)
  - Matricule (int)
  - Téléphone (long)
  - Nombre de cours contributoire (unsigned int)
  - Moyenne générale (float)

10

# COMMENT DÉFINIR UNE STRUCTURE ?

11

## GRAMMAIRE D'UNE DÉFINITION D'UNE STRUCTURE EN C++

- C++ permet la définition directe des structures
- La forme générale

```
struct NomDeLaStructure_s
{
    type NomDuMembre_1;
    type NomDuMembre_2;
    ...
    type NomDuMembre_N;
};
```
- Les types simples (ou d'agrégat) qui forment la structure sont appelés des **membres** de la structure.
- Tous les noms de structure doivent se terminer par **\_s** (convention)

12

## À QUOI RESSEMBLE LA DÉFINITION DE LA STRUCTURE PERSONNE ÉTUDIANTE

### Planification de notre structure

Personne étudiante  
 Nom (string)  
 Prénom (string)  
 Matricule (int)  
 Téléphone (long)  
 Nombre de cours contributoire  
 (unsigned int)  
 Moyenne générale (float)

### Version C++ de notre structure

```
struct PersonneEtudiante_s
{
    string Nom;
    string Prenom;
    int Matricule;
    long Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

13

## OÙ DOIT-ON PLACER LA DÉFINITION

- La définition va toujours avant la fonction *main*
  - Juste après les instructions de préprocesseurs et `using namespace std`
- S'il y en a trop, on place ces définitions dans un fichier d'en-tête (\*.h)
  - Nous verrons dans quelques cours ce que ça signifie

14

## À QUOI RESSEMBLE NOTRE EXEMPLE DE LA PERSONNE ÉTUDIANTE

```
#include <iostream>           int main()
using namespace std;         {
struct PersonneEtudiante_s   ...
{                           ...
    string Nom;
    string Prenom;
    int Matricule;
    long Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

15

## EXISTE-T-IL UNE CONVENTION DE CODAGE ?

Simplement, oui !

16

## 5 RÈGLES DE CODAGE

- Ce ne sont pas des règles du langage. Ce sont des ententes entre les programmeuses et les programmeurs pour faciliter la lecture et la maintenance des logiciels
- Les cinq règles doivent être observées en tout temps
  1. Les structures doivent être déclarées avant la fonction *main*
  2. Lorsqu'il y a plus de cinq structures, ces dernières doivent être écrites dans un fichier d'en-tête (\*.h)
  3. Le nom (identifiant) d'une structure doit toujours se terminer par \_s
  4. Les variables du type de la structure doivent être déclarées dans une fonction soit la fonction *main* ou une autre fonction
  5. Les déclarations de variable de type de la structure au niveau globale devront être justifiées
    - Nous expliquerons ce que ça signifie dans quelques cours

17

## COMMENT UTILISER UNE STRUCTURE ?

18

## PREMIÈREMENT, DÉCLARONS UNE VARIABLE

- La définition d'une structure permet de la décrire l'entité
- Pour l'utiliser, il faut **déclarer** une variable du type de la structure
  - La définition se fait comme toutes les autres types
  - Forme générale :  
`<Type_structure> <identifiant>;`
  - Dans notre exemple :  
`PersonneEtudiante_s LaPersonne;`
- C'est à ce moment que le compilateur procède à l'allocation de l'espace mémoire
  - Maintenant la structure existe dans la RAM
- À ce moment, les valeurs de chaque membre correspondront aux valeurs par défaut

19

## COMMENT SPÉCIFIER UNE VALEUR LORS DE L'INITIALISATION ?

- On peut spécifier les valeurs initiales de la variable avec la même syntaxe que les tableaux
  - La position de la valeur correspond à la position de la déclaration du membre
  - Forme générale  
`<Type_structure> <identifiant> = { <valeur_membre1>, <val_mem2>, ..., <val_memN>};`
  - Dans notre exemple :  
`PersonneEtudiante_s personne = {"Borduas", "Godefroy", 21223366, 5797214447, 2 + 3, 42.0};`
  - L'ordre des valeurs est donc :  
`{Nom, Prenom, Matricule, Telephone, CoursContri, MoyenneGenerale}`

20

## ACCÉDER AUX MEMBRES D'UNE STRUCTURE

- L'accès aux membres permet de :
  - Lire le contenu (sa valeur)
  - Écrire du contenu
- L'accès (lecture/écriture) se fait via l'**opérateur point** (.) placer entre la variable de type de la structure et le membre
  - Forme générale :
  - <variable>.<identifiant du membre>
  - Dans notre exemple :
  - personne.nom
  - Signifie :
  - Nom **élément de** personne

21

## AFFECTER UNE VALEUR AU MEMBRE

- L'affectation se fait grâce à l'opérateur d'affectation (=)
  - Forme générale :
  - <variable>.<identifiant du membre> = <valeur>;
  - Dans notre exemple :
  - personne.Telephone = 5149823437;

22

## LIRE LE CONTENU D'UN MEMBRE

- Pour utiliser le membre (via l'opérateur point) comme tout autre variable

- Prenons l'exemple du nouveau cours

- Premier code sans structure

```
Sn = MoyenneGen * CoursContributoire;
```

- Deuxième code avec structure

```
Sn = personne.MoyenneGenerale * personne.CoursContri;
```

23

REPRENONS L'EXERCICE DU DÉBUT DU COURS

24

MAINTENANT, PENSEZ AU PROGRAMME AVEC 10 PERSONNES ÉTUDIANTES

25

UTILISER CIN OU COUT SUR LES STRUCTURES

26

## CIN ET COUT LISSENT LES TYPES SIMPLES SEULEMENT

- Impossible de lire directement un type de structure
- Doit être appelé sur un membre seulement  
`cin >> personne.Telephone;`  
`cout << personne.MoyenneGenerale << endl;`

27

## L'IMPACT SUR LE DICTIONNAIRE DE DONNÉE

28

14

## ON DÉTAILLE CHAQUE MEMBRE

- Comme les types simples, on doit donner l'identifiant de la variable, la signification du type et son type
- Il en va de même pour les membres
- La valeur est donnée par membre

| Identificateur | Signification                                                    | Type                | Valeur     |
|----------------|------------------------------------------------------------------|---------------------|------------|
| Personne       | Structure représentant les informations d'une personne étudiante | PersonneEtudiante_s |            |
| Nom            | Nom de la personne étudiante                                     | string              | "Borduas"  |
| Prenom         | Prénom de la personne étudiante                                  | string              | "Godefroy" |
| Matricule      | Identifiant numérique de la personne étudiante                   | int                 | 21223366   |
| Telephone      | Numéro de téléphone de la personne étudiante                     | long                | 5797214447 |
| CoursContri    | Nombre de cours contribuant à la moyenne générale                | unsigned int        | 5          |
| MoyenneGeneral | Moyenne générale de la personne étudiante                        | float               | 42.0       |

29

## LES STRUCTURES DANS DES STRUCTURES (INCEPTION)

30

## UNE STRUCTURE PEUT CONTENIR TOUS LES TYPES

- Il est possible d'ajouter un type de structure dans un autre type de structure
- Exemple, le numéro de téléphone est une structure
 

```
Telephone_s
        IdentifiantRegional
        IdentifiantLocal
        IdentifiantUnique
      · Dans le cas du numéro 579 721-4447
      Telephone_s
        IdentifiantRegional = 579
        IdentifiantLocal = 721
        IdentifiantUnique = 4447
```

31

## UNE FOIS CRÉÉ, ON L'AJOUTE DANS LA STRUCTURE

- Après avoir défini la structure imbriquée, on l'ajoute dans notre structure principale :
 

```
struct Telephone
{
    int IdentifiantRegionale;    struct PersonneEtudiante_s
    int IdentifiantLocale;       {
    int IdentifiantUnique;
};

    string Nom;
    string Prenom;
    int Matricule;
    Telephone_s Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

32

## COMMENT ACCÉDER AUX MEMBRES DE LA STRUCTURE IMBRIQUÉE ?

- Il faut passer par le membre contenant la structure imbriquée et ensuite accéder au membre souhaiter
- Toujours utiliser l'opérateur point
- Exemple pour accéder à `IdentifiantRegionale` :  
`personne.Telephone.IdentifiantRegionale`
  - Mode lecture  
`cout << personne.Telephone.IdentifiantRegionale;`
  - Mode écriture  
`personne.Telephone.IdentifiantRegionale = 514;`

33

## UTILISER DES TABLEAUX DE STRUCTURE

34

## C'EST COMME LES TYPES SIMPLES

- Pour créer un tableau d'un type de structure, on utilise la même syntaxe que pour les types simples
  - Forme générale :  
`<type de structure> <identifiant>[<taille>];`
  - Dans notre exemple :  
`PersonneEtudiante_s personnes[10];`
- Par la suite, le compilateur traite les tableaux de structure comme tout autre type de tableau
- Leur fonctionnement est identique

35

## LECTURE ET AFFECTATION DES VALEURS

- Lecture :  
`cout << NomTableau[indice];`
- Affectation :  
`cin >> NomTableau[indice];`

36

# EXERCICES

37

## LES POINTS DANS UN PLAN

- Créer une structure qui représente un point dans un plan
- Demander à la personne utilisatrice d'entrée deux points et calculer sa distance entre ces points
  - Calcul de la distance entre deux points

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

| Identificateur | Signification                                                             | Type       | Valeur     |
|----------------|---------------------------------------------------------------------------|------------|------------|
| Point          | Structure représentant un point dans l'espace                             | Choisissez |            |
| x              | Position réelle (nombre flottant) sur l'axe des abscisses (gauche-droite) | Choisissez | "Borduas"  |
| Y              | Position réelle (nombre flottant) sur l'axe des ordonnées (haut-bas)      | Choisissez | "Godefroy" |

38

19

## FONCTIONS MATHÉMATIQUES UTILES

- Calcul d'une racine carrée
  - Dans la bibliothèque : <cmath>
  - Prototype :
    - double Resultat = sqrt(Valeur);
    - float Resultat = sqrt(Valeur);
  - <https://en.cppreference.com/w/cpp/numeric/math/sqrt>
- Calcul d'une puissance carrée
  - Dans la bibliothèque : <cmath>
  - Prototype :
    - double Resultat = pow(Valeur, 2);
    - float Resultat = pow(Valeur, 2);
  - <https://en.cppreference.com/w/cpp/numeric/math/pow>

39

## MANIPULER DES TABLEAUX D'ÉTUDIANT

- Écrire un programme C, qui lit les noms complets des étudiants et leurs moyennes dans un tableau de structures. Puis actualise ces moyennes en ajoutant un bonus de:
  - 1 point pour les étudiants ayant une note strictement inférieure à 10.
  - 0.5 point pour les étudiants ayant une note entre 10 et 15 inclusive.

*N.B.:* la structure doit avoir deux éléments: une chaîne de caractères et un réel.

40

## LES VILLES ET LES HABITANTS

- Écrire un programme C, qui lit un ensemble de villes avec leur nombre d'habitants dans un tableau de structures, les trie et les affiche dans l'ordre croissant des nombres d'habitants.

| Identificateur | Signification                    | Type       | Valeur     |
|----------------|----------------------------------|------------|------------|
| Ville          | Structure représentant une ville | Choisissez |            |
| Nom            | Nom de la ville                  | Choisissez | "Borduas"  |
| NbreHabitant   | Nombre d'habitant                | Choisissez | "Godefroy" |

41

## UTILISATION DE *TYPEDEF*

Petite info gratuite !

42

## DÉFINITION D'UN TYPE

- Nous avons vu la définition d'une structure

```
struct IdentifiantDeLaStructure { ... };
```
- Dans le langage C++, cette déclaration définit un nouveau type qui se nomme **IdentifiantDeLaStructure**
- Toutefois, ce n'est pas valable en C pur.
- La définition précédente définit seulement une structure en C pur et non un type.
  - Le type est alors **struct IdentifiantDeLaStructure**
- Pour définir le type, l'instruction est alors :

```
Typedef struct IdentifiantDeLaStructure IdentifiantDeLaStructure_s
```

43

## DÉFINIR DIRECTEMENT LA STRUCTURE

- On peut définir directement la structure

```
typedef struct {  
    ...  
} IdentifiantDeLaStructure_s;
```

44

# LES ÉNUMÉRATIONS

Petite info gratuite II

45

## QU'EST-CE QU'UNE ÉNUMÉRATION ?

- Au sens littéraire, l'énumération est une « énonciation successive des éléments d'un tout ».
- L'énumération est donc une liste d'élément.
- Exemple :
  - Couleur : rouge, bleu, jaune et vert
  - Forme : carré, rond, triangle et pentagone
  - Fruit : pomme, fraise, tomate et banane
- En programmation, l'énumération est une liste fixe.
  - Elle ne peut être réduite
  - Elle ne peut être allongée

46

## À QUOI SERVENT LES ÉNUMÉRATIONS ?

- À limiter les choix dans un programme
- À faciliter la gestion des choix
- Imaginez qu'une fonction offre trois choix pour fonctionner :
  - Additionner les paramètres
  - Soustraire les paramètres
  - Multiplier les paramètres
- L'énumération permet de limiter le choix à ces trois options
- La fonction recevra simplement la valeur liée à l'énumération

47

## REPRÉSENTATION D'UNE ÉNUMÉRATION EN C

- L'énumération est un type entier déguisé
- Chaque élément de l'énumération est représenté par un nombre entier
- Pour nous, on utilise une étiquette au lieu du nombre
- Par la suite, on utilise l'énumération comme un type

48

## DÉFINIR ET UTILISER L'ÉNUMÉRATION

- La définition d'une énumération à la forme suivante :
  - Forme générale :

```
enum IdentifiantEnumeration { valeur1, valeur2, ..., valeurN};
```
  - Exemple :

```
enum Couleur {jaune, rouge, bleu, vert};
```
- Affecter une énumération :

```
Couleur c1 = jaune;
```
- Cette action est interdite :

```
jaune = 3;
```

Jaune est une valeur dès quelle est défini dans une énumération

49

## DÉFINIR LA VALEUR DES ÉNUMÉRATIONS

- Comme les énumérations sont des listes d'entier, on peut définir la valeur d'un item
- Pour définir la valeur, on affecte une valeur entière à l'étiquette
- Exemple :

```
enum Couleur {jaune = 5, bleu, vert = 4, bleu = 42};
```

50

## UTILISATION D'UNE ÉNUMÉRATION DANS UN CALCUL ENTIER

- Comme l'énumération est un entier, on peut l'utiliser dans les calculs

```
int p;  
enum Couleur {jaune = 5, bleu, vert = 4, bleu = 42};  
Couleur c = jaune;  
p = 2 + vert * jaune;
```

- La valeur de p est alors 22, car vert = 4 et jaune = 5

# Amusons-nous... la surcharge d'opérateur

420-C21-IN PROGRAMMATION II

GODEFROY BORDUAS – AUTOMNE 2021

MODULE BONUS

# Comment définir des opérateurs

- ▶ Les opérateurs (<<, +, -=, etc.) sont, en C++, des fonctions définies pour les types qui existent de base dans le langage
- ▶ Comme les structures sont des types définis hors du langage, il n'existe pas de fonction définie pour elles
- ▶ Il est nécessaire de définir nos fonctions d'opérateur. Nous parlons alors de **surcharge d'opérateur**
- ▶ Prenons les opérateurs << et >>, ces derniers agissent sur les flux d'entrée et de sortie. Or, il existe deux types de flux `istream` et `ostream`. Il s'agit des flux de base
  - ▶ Sortie (lié à >>) : son type est `ostream` (`cout` spécialise `ostream` pour la console)
  - ▶ Entrée (lié à <<) : son type est `istream` (`cin` spécialise `istream` pour la console)

# Une syntaxe simple à une idée complexe

- ▶ Ici, nous présentons l'idée pour l'opérateur <<. Toutefois, le principe reste valide pour tous les opérateurs.
  - ▶ Néanmoins, il faut faire attention aux types de paramètres impliqués
- ▶ Le prototype (générale) de la fonction d'opérateur << sera :

```
void operator<<(std::ostream&, const Structure_s);
```
- ▶ Pour tous les opérateurs, le nom de la fonction est toujours **operator** suivi du symbole de l'opérateur
- ▶ Le symbole & désigne une référence. Il s'agit d'un pointeur autogéré par le compilateur.
  - ▶ Ainsi le formalisme ne s'applique pas
- ▶ Nous appliquons la condition **const** sur le paramètre du flux pour éviter de le modifier accidentellement

# Qu'est-ce que les deux paramètres signifient ?

```
void operator<<(std::ostream&, const Structure_s);
```

- ▶ Le premier reçoit le flux à manipuler
- ▶ Le second reçoit l'entité à utiliser dans la manipulation
- ▶ En gros, prenez la ligne suivante :

```
cout << 5;
```

- ▶ La fonction d'opérateur appeler a donc la signature suivante :

```
void operator<<(std::ostream&, const Int);
```

- ▶ La première ligne peut être remplacée par :

```
operator<<(cout, 5);
```

# Imaginons la structure Vecteur2

5

- ▶ Regardez le code **vecteur2.cpp**

```
#include <iostream>

struct Vecteur2_s { float x; float y; };

void operator<<(std::ostream&, const Vecteur2_s&);

void operator<<(std::ostream& stream, const Vecteur2_s& vec) {
    stream << "(" << vec.x << ", " << vec.y << ")";
}
```

# Petit problème, essayer d'ajouter une fin de ligne après l'appel

- ▶ Avec les choses actuelles, vous ne pouvez pas appliquer `endl` après avoir appelé `cout << vec`
- ▶ Ceci vient du fait que votre fonction d'opérateur ne retourne pas de flux.
- ▶ Regardons l'exemple suivant :

```
cout << vecteur << endl;
```

- ▶ Ceci revient à dire que (on applique la même logique qu'avant) :  
`operator<<(operator<<(cout, vecteur), endl);`
- ▶ Pour avoir ce résultat, il faut utiliser le prototype suivant :  
`std::ostream& operator<<(std::ostream&, const Vecteur2_s&);`

# Notre exemple précédent devient

7

- ▶ Regardez le code **vecteur2\_plus.cpp**

```
#include <iostream>

struct Vecteur2_s { float x; float y; };

std::ostream& operator<<(std::ostream&, const Vecteur2_s&);

std::ostream& operator<<(std::ostream& stream, const Vecteur2_s& vec) {
    stream << "(" << vec.x << ", " << vec.y << ")";
    return stream;
}
```

# Tous les opérateurs peuvent être surchargés

- ▶ Regardez le code **vecteur2\_plusplus.cpp**
  - ▶ Codez les fonctions suivantes :

```
void operator+=(Vecteur2_s&, const Vecteur2_s&);  
void operator-=(Vecteur2_s&, const Vecteur2_s&);  
void operator*=(Vecteur2_s&, const float&);
```

```
Vecteur2_s operator+(const Vecteur2_s&, const Vecteur2_s&);  
Vecteur2_s operator-(const Vecteur2_s&, const Vecteur2_s&);  
Vecteur2_s operator*(const Vecteur2_s&, const float&);
```

# Allons-y aussi avec les binaires

- ▶ Regardez le code **vecteur2\_plusplus\_bin.cpp**
  - ▶ Codez les fonctions suivantes :

```
bool operator==(const Vecteur2_s&, const Vecteur2_s&);  
bool operator!=(const Vecteur2_s&, const Vecteur2_s&);
```

# Exercice : Les nombres rationnels

- ▶ Créer une structure Rationnel (deux valeurs entières : numérateur et dénominateur)
- ▶ Définissez les opérateurs : +, -, \*, /, <<
- ▶ Petit rappel mathématique :

$$\frac{p}{q} + \frac{r}{s} = \frac{s * p + r * q}{q * s}$$

$$\frac{p}{q} * \frac{r}{s} = \frac{p * r}{q * s}$$

$$\frac{p}{q} \div \frac{r}{s} = \frac{p}{q} * \frac{s}{r}$$

# LES PROGRAMMES MODULAIRES DIVISER SON PROGRAMME EN FONCTIONS

Module 3  
420-C21-IN Programmation II  
Godefroy Borduas – Automne 2021

1

## QUEL EST L'OBJECTIF DU COURS ?

- Comprendre la notion de fonction
  - Utilité d'une fonction
- Comment créer une fonction ?
  - Définition des fonctions (prototype)
  - Déclaration des fonctions (corps)
- Comment utiliser une fonction ?
- Comment partager des informations avec les fonctions ?
- Petite attention au passage des paramètres et à la portée des variables
- Qu'est-ce qu'une bonne fonction ?

2

## LA NOTION DE FONCTION

3

## QU'EST-CE QU'UNE FONCTION ?

- Petit bout de code
- Réalise une opération concrète et définie
- Exemple :
  - Calculer la puissance de  $x$  par  $y$
  - Calculer la différence de jour entre deux dates
  - Gérer un menu

4

2

## POURQUOI UTILISER UNE FONCTION ?

- Réduire la taille du *main*
- Faciliter la lecture et la révision du code
- Réduis l'écriture de code

5

## ALLONS-Y AVEC UN EXEMPLE DE PROGRAMME

- Créer un programme qui réalise le calcul  $x^y$
- L'algorithme naïf est :
 

```
Resultat = 1
Pour i = 1, y, 1
    Resultat = Resultat * x
Écrire 'La puissance de' x 'est' Resultat
```
- Votre programme doit réaliser les étapes suivantes :
 

```
Calculer 316 (x = 3, y = 16)
Calculer 4224 (x = 42, y = 24)
Calculer 10482048 (x = 1048, y = 2048)
```

**INTERDIT !**  
D'utiliser la  
méthode pow

6

## QUELLE EST VOTRE CONCLUSION ?

- C'est long
- C'est répétitif
- Plusieurs sources d'erreur possible
  - Oublier de réinitialiser la variable Résultat
  - Oublier de spécifier la valeur x ou de y
  - Se tromper dans les variables
- C'est long pour corriger une erreur

7

## LA SOLUTION

- Créer une fonction qui calcule la puissance
- Elle pourra être appelée à chaque besoin
- Les variables dans la fonction sont réinitialisées à chaque fois
- Besoin de changer ou de corriger ? Un seul endroit
- permet de partager sa méthode
  - On verra ce point plus tard

8

## CRÉER SES FONCTIONS

Première étape : Déclarer sa fonction

9

## QU'EST-CE QUE LA DÉCLARATION DE FONCTION ?

- Une annonce pour le compilateur
- Le compilateur réserve l'espace mémoire
  - Réserve le nom
  - Réserve l'espace des paramètres (variable d'entrée)
- S'appelle aussi un **prototype**

10

## COMMENT ÉCRIRE UN PROTOTYPE

- Possède trois champs
  - Son type de retour (ce que la méthode renvoie)
  - Son nom (comment la différencier)
  - Les paramètres de la fonction (ce qu'elle reçoit pour fonctionner)
- Le prototype a toujours la même forme
 

```
Type_de_retour nom_fonction(type para_1, type para_2, ..., type para_N);
```

  - Le nom des paramètres est facultatif, seuls les types comptent
  - Le type **void** indique qu'aucune valeur ne sera renournée. Le retour est donc vide.
- Les prototypes sont toujours décrits **AVANT** le *main*

11

## IMAGINONS LA MÉTHODE PUISSANCE

- Reprenons l'exemple du calcul de la puissance
- Avant d'écrire le prototype, il faut décrire son diagramme d'action
- La fonction d'un diagramme pour une méthode sera toujours comme suit :
 

```
Type_retour Nom(paramètre1, paramètre2, ..., paramètreN)
          Instructions
          retour Variable renournée
```
- Si la fonction n'a pas de paramètre d'entrée, on inscrit simple **void** dans les parenthèses

12

## LE DIAGRAMME D'ACTION ET LA DÉCLARATION DE PUISSANCE

- En somme, le diagramme est :

```
double Puissance (x,y)
    Resultat = 1
    Pour i = 1, y, 1
        Resultat = Resultat * x
    retour Résultat
```

- Son prototype est donc :

```
double Puissance(int x, int y);
```

- Le prototype suivant est aussi valide :

```
double Puissance(int, int);
```

13

## CRÉER SES FONCTIONS

Deuxième étape : Déclarer ses fonctions

14

## SANS CORPS, UNE FONCTION N'EST PAS UTILISABLE

- La déclaration permet de créer la substance de la fonction
- On définit ses instructions (ce qu'elle doit faire)
- Une fonction sans corps n'est pas une fonction
- Les définitions sont toujours placées après le *main*
- Les fonctions ont toujours la même forme :

```
Type_de_retour nom_fonction(type para_1, type para_2, ..., type para_N)
{
    // Série d'instructions
    return Variable_a_retourner;    // Instruction de retour
                                    // (seulement pour les fonctions qui n'ont pas un type void)
}
```

15

## UN LIEN ENTRE LA DÉFINITION ET LA PREMIÈRE LIGNE DE LA DÉCLARATION ?

- Certainement, la définition annonce la déclaration
- Les deux doivent correspondre
  - Le nom des paramètres doit être identique s'ils sont présents dans la définition
- Si la définition ne correspond pas à la première ligne de la déclaration
  - Le compilateur va refuser votre code
  - Le projet ne compilera pas

16

## EXEMPLE DE LA MÉTHODE PUISSANCE

- Pour rappel, la déclaration est :

```
double Puissance(int x, int y);
```
- Par conséquent, sa définition est :

```
double Puissance(int x, int y)
{
    // Liste d'instruction
    return Resultat;
}
```
- Qu'est-ce qu'on met comme instruction ?

17

## LES INSTRUCTIONS SERONT CELLES DE NOTRE DIAGRAMME

```
double Puissance(int x, int y)
{
    double Resultat = 1;
    for(int i = 1; i <= y; i++)
    {
        Resultat *= x;
    }
    return Resultat;
}
```

**Note sur l'instruction de retour**  
*Si l'instruction return n'a aucune valeur, alors elle met fin à la fonction sans rien retournée (valide pour les fonctions de type void).*

18

## UTILISER SES FONCTIONS

En gros... comme toutes les fonctions

19

## L'APPEL D'UNE FONCTION EST SIMPLE

- Il suffit d'utiliser le nom de la fonction suivi des parenthèses

- Exemple d'une fonction sans paramètre

```
void exemple1(); // Définition  
exemple1(); // Utilisation
```

- Exemple d'une fonction avec paramètre

```
void exemple2(int, int); // Définition  
exemple2(42, 42); // Utilisation
```

- Si la fonction a un retour, on affecte la valeur à une fonction

```
int exemple3(int); // Définition  
int n = exemple3(42); // Utilisation
```

### Forme générale

```
nom_fonction(para1, para2, ..., paraN);
```

20

## VOUS AVEZ DÉJÀ UTILISÉ DES FONCTIONS

- Regardez vos notes de C11

```
clrscr();
Rep = _getche();
Rep = MessageBoxA(NULL, "Voulez-vous continuer ?", "Système B11",
MB_YESNO);
```

21

## DANS NOTRE CAS, NOTRE PROGRAMME DEVIENT...

```
#include <iostream>
using namespace std;
double Puissance(int, int);
int main()
{
    double p1 = Puissance(3, 16);
    cout << p1 << endl;
    double p2 = Puissance(42, 24);
    cout << p2 << endl;
    double p3 = Puissance(1024, 2048);
```

```
        cout << p3 << endl;
    }
    double Puissance(int x, int y)
    {
        double Resultat = 1;
        for(int i = 1; i <= y; i++)
        {
            Resultat *= x;
        }
        return Resultat;
    }
```

22

## COMMENT PARTAGER DES INFORMATIONS AVEC LES FONCTIONS ?

23

### PETIT QUIZ RAPIDE !

- Combien de paramètres peut-on transmettre à une fonction ?
  - 256 (est-ce qu'on va vraiment l'atteindre ?)
- Quel type puis-je transmettre dans une fonction ?
  - Tous les types incluant les types simples (int, char, float, ...), les types d'agrégats (string, int[], float[], ...), les types de structure et les pointeurs (notion du prochain cours)
- Combien de variable peut-il être renvoyé par une fonction ?
  - Une seule variable et le type doivent être définis à l'avance
- Quel type peut-être renvoyé par une fonction ?
  - Tous les types incluant les types simples (int, char, float, ...), les types d'agrégats (string, int[], float[], ...), les types de structure et les pointeurs (notion du prochain cours)

24

## PETITE ATTENTION AU PASSAGE DES PARAMÈTRES ET À LA PORTÉE DES VARIABLES

25

## COMMENT C++ TRANSFÈRE VOS VARIABLES ?

- C++ utilise le **passage par valeur**
- En gros, la variable en paramètre est **copiée** dans une nouvelle variable dite **locale**
  - Conséquence : Tous les calculs réalisés sur la variable locale n'affectent pas à la variable d'origine (celle qui a servi à l'appel de la fonction)
  - Conséquence II : Le programme doit utiliser le double d'espace pour la même valeur

26

## QU'EST-CE QU'UNE VARIABLE LOCALE?

- Il s'agit d'une variable qui existe uniquement dans son **bloc de déclaration**
- Qu'est-ce qu'un **bloc de déclaration** ?
  - Il s'agit d'une suite d'instruction incluse entre des accolades {}.
  - Une fonction est un bloc de déclaration
  - La liste des instructions de if est un autre exemple de bloc
- En somme, la variable locale existe tant et aussi longtemps que nous sommes dans le bloc
  - Dès qu'on sort, la variable est détruite

27

## EFFET DU PASSAGE DE VARIABLE

```
void Test(int);
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
```

Variable locale à Main

```
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

- Variable locale à Test
- Affiche la valeur 9
  - Affiche la valeur 2

28

14

## IMPACT SUR LA MÉMOIRE

- Après la première ligne de main

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
```

| Adresse | Valeur |
|---------|--------|
| 0x...1  |        |
| 0x...2  |        |
| 0x...3  |        |
| 0x...4  |        |
| 0x...5  |        |
| 0x...6  |        |
| 0x...7  |        |
| 0x...8  |        |
| 0x...9  |        |

29

## IMPACT SUR LA MÉMOIRE

- Après la **deuxième** ligne de main

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

| Adresse | Valeur |
|---------|--------|
| 0x...1  |        |
| 0x...2  |        |
| 0x...3  |        |
| 0x...4  |        |
| 0x...5  |        |
| 0x...6  |        |
| 0x...7  |        |
| 0x...8  |        |
| 0x...9  |        |

30

## IMPACT SUR LA MÉMOIRE

- Après la **première** ligne de **test**

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

| Adresse | Valeur                     |
|---------|----------------------------|
| 0x...1  | <b>int</b><br><b>i (2)</b> |
| 0x...2  |                            |
| 0x...3  |                            |
| 0x...4  |                            |
| 0x...5  |                            |
| 0x...6  | <b>int</b><br><b>j (9)</b> |
| 0x...7  |                            |
| 0x...8  |                            |
| 0x...9  |                            |

31

## IMPACT SUR LA MÉMOIRE

- Après la **troisième** ligne de **main**

```
int main() {
    int i = 2;
    Test(i);
    cout << i << endl;
}
void Test(int j) {
    j = 9;
    cout << j << endl;
    return;
}
```

| Adresse | Valeur                     |
|---------|----------------------------|
| 0x...1  | <b>int</b><br><b>i (2)</b> |
| 0x...2  |                            |
| 0x...3  |                            |
| 0x...4  |                            |
| 0x...5  |                            |
| 0x...6  |                            |
| 0x...7  |                            |
| 0x...8  |                            |
| 0x...9  |                            |

32

MAINTENANT, IMAGINER UN TABLEAU DE  
STRING DE 256 CASES À TRANSFÉRER

**Spoiler !**  
On voit une solution dès le  
prochain cours.

33

QU'EST-CE QU'UNE BONNE  
FONCTION ?

34

## UNE BONNE FONCTION RESPECTE LE CLEAN CODE

1. Une fonction doit être courte
  - Maximum de 25 lignes de codes, si on en a besoin de plus, alors on parle d'une autre fonction.
2. La fonction n'a qu'une seule responsabilité et ne fait qu'une chose
  - La fonction ne doit faire qu'une seule tâche et non deux
3. Le nom de la fonction doit correspondre à ce que la fonction fait
  - Imaginer la fonction calcul avec le nom **Babloubabloblou**
4. Le bloc et les instructions doivent être indentés
  - Pitié mes yeux
5. Aucune duplication de code
  - Pourquoi se répéter ? On est des paresseux
6. Aucun effet secondaire (side effect)
  - La fonction ne doit pas modifier directement une valeur
7. Si c'est mort, ça dégage !
  - Ce n'est pas beau et pas lisible

C'est un critère  
d'évaluation !

35

## SURPRISE ! LA FONCTION RÉCURSIVE

36

18

## QU'EST-CE QU'UNE FONCTION RÉCURSIVE ?

- Fonction dont le calcul nécessite l'appel d'elle-même
- Exemple : La suite de Fibonacci
  - Suite mathématique dont les valeurs dépendent des valeurs précédentes
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ...
  - Défini mathématiquement par :
$$F_n = \begin{cases} 0, & \text{si } n = 0 \\ 1, & \text{si } n = 1 \\ F_{n-1} + F_{n-2}, & \text{si } n > 1 \end{cases}$$
- Les fonctions récursives ont toujours un **point d'arrêt**
  - Sinon, on tombe dans une boucle infinie

37

## EN TERMES DE CODE :

- Besoin d'un paramètre : **n** soit l'indice du nombre à calculer
- Retour la valeur de la suite à la position **n**
- La suite est décrite pour les entiers positifs seulement
- Le prototype est alors :
 

```
unsigned int Fibonacci(unsigned int);
```
- Le calcul est donc :
 

```
Si n = 0, retourne 0
Si n = 1, retourne 1
Sinon retourne F(n - 1) + F(n - 2)
```

<https://replit.com/@ColonelSaumon/Fibonacci>

38

19

## QU'EST-CE QUI SE PASSE DANS UNE FONCTION SANS POINT D'ARRÊT

- La suite infinie de Godefroy

$$g_i = \begin{cases} \frac{n_{i-1}}{2}, & \text{si } n \text{ est pair} \\ 2n_{i-1}, & \text{si } n \text{ est impair} \\ 1000, & \text{si } n = 2 \\ -n_{i-1}, & \text{si } n < 0 \\ 500, & \text{si } n = 0 \end{cases}$$

- La suite ne peut jamais donner de résultat

<https://replit.com/@ColonelSaumon/SuiteInfiniDeGodefroy>

39

## EXERCICES

40

20

## SÉRIE D'EXERCICES

1. Écrivez une fonction **distance** ayant comme paramètres 4 doubles  $xa,ya$  et  $xb,yb$  qui représentent les coordonnées de deux points A et B et qui renvoient la distance AB. Tester cette fonction.
2. Reprenez la fonction précédente, mais avec la structure **Point**
3. Écrivez une fonction  $f$  ayant en paramètres un tableau  $t$  de taille quelconque et un entier  $n$  indiquant la taille du tableau.  $f$  doit renvoyer par un booléen indiquant s'il existe une valeur comprise entre 0 et 10 dans les  $n$  premières cases du tableau  $t$ . Tester cette fonction.

41

## EXERCICE DE FONCTION RÉCURSIVE

1. Écrivez une fonction qui retourne la valeur de la factorielle de l'entier  $n$  transmit par paramètre.
  - La factorielle correspond à la multiplication de toutes les valeurs entiers entre 0 et  $n$ .
  - La factorielle de 0 est par définition 1.
2. Écrivez une fonction qui calcule la grande suite de Godefroy. Votre fonction doit retourner la valeur pour l'indice  $n$ .  $n$  est un entier transmis par paramètre à la fonction.
  - La grande suite de Godefroy est décrite comme suit :
$$G_n = \begin{cases} G_{-n}, & n < 0 \\ 0, & si \ n = 0 \\ i + 1, & si \ 1 \leq n \leq 9 \\ G_{i-1} + G_{i-2} + 2G_{i-3}, & si \ n \geq 10 \end{cases}$$

42

# LA CRÉATION DE SES PROPRES TYPES LES STRUCTURES

Module 2

420-C21-IN Programmation II

Godefroy Borduas – Automne 2021

1

## PLAN DE LEÇON

- Qu'est-ce qu'une structure ?
- Comment définir une structure ?
- Existe-t-il une convention de codage ?
- Comment utiliser une structure ?
- Utiliser cin ou cout sur les structures
- L'impact sur le dictionnaire de donnée
- Les structures dans des structures (Inception)
- Utiliser des tableaux de structure
- Exercices

2

# QU'EST-CE QU'UNE STRUCTURE ?

3

## ÇA TOUCHE LA MÉMOIRE

- Un des problèmes importants est l'organisation de l'information
- En C11, vous avez vu les types « simples » et les types « agrégats »
  - Type simple : Type de variable de base (nombre, caractères)
  - Type agrégats : Type réunissant plusieurs valeurs de type simple (tableau, string)
- Très utile pour des problèmes simples, exemple :
  - Calcul sur des entiers (trois entiers)
  - Collecter les notes d'une classe (tableau de notes)

4

2

## MAINTENANT, PRENONS UNE INFORMATION PLUS COMPLEXE

- Le profil d'une personne étudiante :
  - Nom
  - Prénom
  - Matricule
  - Téléphone
  - Nombre de cours contributoire
  - Moyenne générale
- Maintenant, créez un programme informatique afin de gérer le profil pour une personne étudiante
  - Modifier le téléphone
  - Modifier la moyenne générale

5

QU'AVEZ-VOUS REMARQUÉ ?

6

3

## ALLONS-Y AVEC UNE EXPÉRIENCE DE PENSÉE

- Imaginez que le système doit gérer, en tout temps, dix personnes étudiantes
- Comment allez-vous modifier le code ?
- Qu'est-ce que vous en concluez ?
- Intuitivement, y a-t-il une solution ?

7

## PENSONS EN TERMES D'ENTITÉ

- Et si la personne étudiante était une entité informatique
  - Entité : Élément informatique manipulable et traitable (ex. int, float, etc.)
- Dans ce cas, la personne étudiante aura des **sous-entités**
- En langage informatique, on parle de **structure**

8

## DÉFINITION

Une **structure** est un type informatique composé ou dérivé des types simples. Ce type permet de manipuler un ensemble d'information comme un seul objet.

Similaire à un tableau, la structure réunit au même endroit toutes les informations relatives à une même entité.

Une fois créée, la structure devient un type C++ comme les int, les floats et compagnie.

9

## À RESSEMBLE UNE STRUCTURE

- Revenons à la personne étudiante, il est possible de créer une structure pour réunir toutes les informations au même endroit.
- La structure ressemble à ceci  
Personne étudiante
  - Nom (string)
  - Prénom (string)
  - Matricule (int)
  - Téléphone (long)
  - Nombre de cours contributoire (unsigned int)
  - Moyenne générale (float)

10

# COMMENT DÉFINIR UNE STRUCTURE ?

11

## GRAMMAIRE D'UNE DÉFINITION D'UNE STRUCTURE EN C++

- C++ permet la définition directe des structures
- La forme générale

```
struct NomDeLaStructure_s
{
    type NomDuMembre_1;
    type NomDuMembre_2;
    ...
    type NomDuMembre_N;
};
```
- Les types simples (ou d'agrégat) qui forment la structure sont appelés des **membres** de la structure.
- Tous les noms de structure doivent se terminer par **\_s** (convention)

12

## À QUOI RESSEMBLE LA DÉFINITION DE LA STRUCTURE PERSONNE ÉTUDIANTE

### Planification de notre structure

Personne étudiante  
 Nom (string)  
 Prénom (string)  
 Matricule (int)  
 Téléphone (long)  
 Nombre de cours contributoire  
 (unsigned int)  
 Moyenne générale (float)

### Version C++ de notre structure

```
struct PersonneEtudiante_s
{
    string Nom;
    string Prenom;
    int Matricule;
    long Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

13

## OÙ DOIT-ON PLACER LA DÉFINITION

- La définition va toujours avant la fonction *main*
  - Juste après les instructions de préprocesseurs et `using namespace std`
- S'il y en a trop, on place ces définitions dans un fichier d'en-tête (\*.h)
  - Nous verrons dans quelques cours ce que ça signifie

14

## À QUOI RESSEMBLE NOTRE EXEMPLE DE LA PERSONNE ÉTUDIANTE

```
#include <iostream>           int main()
using namespace std;         {
struct PersonneEtudiante_s   ...
{                           ...
    string Nom;
    string Prenom;
    int Matricule;
    long Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

15

## EXISTE-T-IL UNE CONVENTION DE CODAGE ?

Simplement, oui !

16

## 5 RÈGLES DE CODAGE

- Ce ne sont pas des règles du langage. Ce sont des ententes entre les programmeuses et les programmeurs pour faciliter la lecture et la maintenance des logiciels
- Les cinq règles doivent être observées en tout temps
  1. Les structures doivent être déclarées avant la fonction *main*
  2. Lorsqu'il y a plus de cinq structures, ces dernières doivent être écrites dans un fichier d'en-tête (\*.h)
  3. Le nom (identifiant) d'une structure doit toujours se terminer par \_s
  4. Les variables du type de la structure doivent être déclarées dans une fonction soit la fonction *main* ou une autre fonction
  5. Les déclarations de variable de type de la structure au niveau globale devront être justifiées
    - Nous expliquerons ce que ça signifie dans quelques cours

17

## COMMENT UTILISER UNE STRUCTURE ?

18

## PREMIÈREMENT, DÉCLARONS UNE VARIABLE

- La définition d'une structure permet de la décrire l'entité
- Pour l'utiliser, il faut **déclarer** une variable du type de la structure
  - La définition se fait comme toutes les autres types
  - Forme générale :  
`<Type_structure> <identifiant>;`
  - Dans notre exemple :  
`PersonneEtudiante_s LaPersonne;`
- C'est à ce moment que le compilateur procède à l'allocation de l'espace mémoire
  - Maintenant la structure existe dans la RAM
- À ce moment, les valeurs de chaque membre correspondront aux valeurs par défaut

19

## COMMENT SPÉCIFIER UNE VALEUR LORS DE L'INITIALISATION ?

- On peut spécifier les valeurs initiales de la variable avec la même syntaxe que les tableaux
  - La position de la valeur correspond à la position de la déclaration du membre
  - Forme générale  
`<Type_structure> <identifiant> = { <valeur_membre1>, <val_mem2>, ..., <val_memN>};`
  - Dans notre exemple :  
`PersonneEtudiante_s personne = {"Borduas", "Godefroy", 21223366, 5797214447, 2 + 3, 42.0};`
  - L'ordre des valeurs est donc :  
`{Nom, Prenom, Matricule, Telephone, CoursContri, MoyenneGenerale}`

20

## ACCÉDER AUX MEMBRES D'UNE STRUCTURE

- L'accès aux membres permet de :
  - Lire le contenu (sa valeur)
  - Écrire du contenu
- L'accès (lecture/écriture) se fait via l'**opérateur point** (.) placer entre la variable de type de la structure et le membre
  - Forme générale :
  - <variable>.<identifiant du membre>
  - Dans notre exemple :
  - personne.nom
  - Signifie :
  - Nom **élément de** personne

21

## AFFECTER UNE VALEUR AU MEMBRE

- L'affectation se fait grâce à l'opérateur d'affectation (=)
  - Forme générale :
  - <variable>.<identifiant du membre> = <valeur>;
  - Dans notre exemple :
  - personne.Telephone = 5149823437;

22

## LIRE LE CONTENU D'UN MEMBRE

- Pour utiliser le membre (via l'opérateur point) comme tout autre variable

- Prenons l'exemple du nouveau cours

- Premier code sans structure

```
Sn = MoyenneGen * CoursContributoire;
```

- Deuxième code avec structure

```
Sn = personne.MoyenneGenerale * personne.CoursContri;
```

23

REPRENONS L'EXERCICE DU DÉBUT DU COURS

24

MAINTENANT, PENSEZ AU PROGRAMME AVEC 10 PERSONNES ÉTUDIANTES

25

UTILISER CIN OU COUT SUR LES STRUCTURES

26

## CIN ET COUT LISSENT LES TYPES SIMPLES SEULEMENT

- Impossible de lire directement un type de structure
- Doit être appelé sur un membre seulement  
`cin >> personne.Telephone;`  
`cout << personne.MoyenneGenerale << endl;`

27

## L'IMPACT SUR LE DICTIONNAIRE DE DONNÉE

28

14

## ON DÉTAILLE CHAQUE MEMBRE

- Comme les types simples, on doit donner l'identifiant de la variable, la signification du type et son type
- Il en va de même pour les membres
- La valeur est donnée par membre

| Identificateur | Signification                                                    | Type                | Valeur     |
|----------------|------------------------------------------------------------------|---------------------|------------|
| Personne       | Structure représentant les informations d'une personne étudiante | PersonneEtudiante_s |            |
| Nom            | Nom de la personne étudiante                                     | string              | "Borduas"  |
| Prenom         | Prénom de la personne étudiante                                  | string              | "Godefroy" |
| Matricule      | Identifiant numérique de la personne étudiante                   | int                 | 21223366   |
| Telephone      | Numéro de téléphone de la personne étudiante                     | long                | 5797214447 |
| CoursContri    | Nombre de cours contribuant à la moyenne générale                | unsigned int        | 5          |
| MoyenneGeneral | Moyenne générale de la personne étudiante                        | float               | 42.0       |

29

## LES STRUCTURES DANS DES STRUCTURES (INCEPTION)

30

## UNE STRUCTURE PEUT CONTENIR TOUS LES TYPES

- Il est possible d'ajouter un type de structure dans un autre type de structure
- Exemple, le numéro de téléphone est une structure
 

```
Telephone_s
        IdentifiantRegional
        IdentifiantLocal
        IdentifiantUnique
      · Dans le cas du numéro 579 721-4447
      Telephone_s
        IdentifiantRegional = 579
        IdentifiantLocal = 721
        IdentifiantUnique = 4447
```

31

## UNE FOIS CRÉÉ, ON L'AJOUTE DANS LA STRUCTURE

- Après avoir défini la structure imbriquée, on l'ajoute dans notre structure principale :
 

```
struct Telephone
{
    int IdentifiantRegionale;    struct PersonneEtudiante_s
    int IdentifiantLocale;       {
    int IdentifiantUnique;
};

    string Nom;
    string Prenom;
    int Matricule;
    Telephone_s Telephone;
    unsigned int CoursContri;
    float MoyenneGenerale;
};
```

32

## COMMENT ACCÉDER AUX MEMBRES DE LA STRUCTURE IMBRIQUÉE ?

- Il faut passer par le membre contenant la structure imbriquée et ensuite accéder au membre souhaiter
- Toujours utiliser l'opérateur point
- Exemple pour accéder à `IdentifiantRegionale` :  
`personne.Telephone.IdentifiantRegionale`
  - Mode lecture  
`cout << personne.Telephone.IdentifiantRegionale;`
  - Mode écriture  
`personne.Telephone.IdentifiantRegionale = 514;`

33

## UTILISER DES TABLEAUX DE STRUCTURE

34

## C'EST COMME LES TYPES SIMPLES

- Pour créer un tableau d'un type de structure, on utilise la même syntaxe que pour les types simples
  - Forme générale :  
`<type de structure> <identifiant>[<taille>];`
  - Dans notre exemple :  
`PersonneEtudiante_s personnes[10];`
- Par la suite, le compilateur traite les tableaux de structure comme tout autre type de tableau
- Leur fonctionnement est identique

35

## LECTURE ET AFFECTATION DES VALEURS

- Lecture :  
`cout << NomTableau[indice];`
- Affectation :  
`cin >> NomTableau[indice];`

36

# EXERCICES

37

## LES POINTS DANS UN PLAN

- Créer une structure qui représente un point dans un plan
- Demander à la personne utilisatrice d'entrée deux points et calculer sa distance entre ces points
  - Calcul de la distance entre deux points

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

| Identificateur | Signification                                                             | Type       | Valeur     |
|----------------|---------------------------------------------------------------------------|------------|------------|
| Point          | Structure représentant un point dans l'espace                             | Choisissez |            |
| x              | Position réelle (nombre flottant) sur l'axe des abscisses (gauche-droite) | Choisissez | "Borduas"  |
| Y              | Position réelle (nombre flottant) sur l'axe des ordonnées (haut-bas)      | Choisissez | "Godefroy" |

38

19

## FONCTIONS MATHÉMATIQUES UTILES

- Calcul d'une racine carrée
  - Dans la bibliothèque : <cmath>
  - Prototype :
    - double Resultat = sqrt(Valeur);
    - float Resultat = sqrt(Valeur);
  - <https://en.cppreference.com/w/cpp/numeric/math/sqrt>
- Calcul d'une puissance carrée
  - Dans la bibliothèque : <cmath>
  - Prototype :
    - double Resultat = pow(Valeur, 2);
    - float Resultat = pow(Valeur, 2);
  - <https://en.cppreference.com/w/cpp/numeric/math/pow>

39

## MANIPULER DES TABLEAUX D'ÉTUDIANT

- Écrire un programme C, qui lit les noms complets des étudiants et leurs moyennes dans un tableau de structures. Puis actualise ces moyennes en ajoutant un bonus de:
  - 1 point pour les étudiants ayant une note strictement inférieure à 10.
  - 0.5 point pour les étudiants ayant une note entre 10 et 15 inclusive.

*N.B.:* la structure doit avoir deux éléments: une chaîne de caractères et un réel.

40

## LES VILLES ET LES HABITANTS

- Écrire un programme C, qui lit un ensemble de villes avec leur nombre d'habitants dans un tableau de structures, les trie et les affiche dans l'ordre croissant des nombres d'habitants.

| Identificateur | Signification                    | Type       | Valeur     |
|----------------|----------------------------------|------------|------------|
| Ville          | Structure représentant une ville | Choisissez |            |
| Nom            | Nom de la ville                  | Choisissez | "Borduas"  |
| NbreHabitant   | Nombre d'habitant                | Choisissez | "Godefroy" |

41

## UTILISATION DE *TYPEDEF*

Petite info gratuite !

42

## DÉFINITION D'UN TYPE

- Nous avons vu la définition d'une structure

```
struct IdentifiantDeLaStructure { ... };
```
- Dans le langage C++, cette déclaration définit un nouveau type qui se nomme **IdentifiantDeLaStructure**
- Toutefois, ce n'est pas valable en C pur.
- La définition précédente définit seulement une structure en C pur et non un type.
  - Le type est alors **struct IdentifiantDeLaStructure**
- Pour définir le type, l'instruction est alors :

```
Typedef struct IdentifiantDeLaStructure IdentifiantDeLaStructure_s
```

43

## DÉFINIR DIRECTEMENT LA STRUCTURE

- On peut définir directement la structure

```
typedef struct {  
    ...  
} IdentifiantDeLaStructure_s;
```

44

# LES ÉNUMÉRATIONS

Petite info gratuite II

45

## QU'EST-CE QU'UNE ÉNUMÉRATION ?

- Au sens littéraire, l'énumération est une « énonciation successive des éléments d'un tout ».
- L'énumération est donc une liste d'élément.
- Exemple :
  - Couleur : rouge, bleu, jaune et vert
  - Forme : carré, rond, triangle et pentagone
  - Fruit : pomme, fraise, tomate et banane
- En programmation, l'énumération est une liste fixe.
  - Elle ne peut être réduite
  - Elle ne peut être allongée

46

## À QUOI SERVENT LES ÉNUMÉRATIONS ?

- À limiter les choix dans un programme
- À faciliter la gestion des choix
- Imaginez qu'une fonction offre trois choix pour fonctionner :
  - Additionner les paramètres
  - Soustraire les paramètres
  - Multiplier les paramètres
- L'énumération permet de limiter le choix à ces trois options
- La fonction recevra simplement la valeur liée à l'énumération

47

## REPRÉSENTATION D'UNE ÉNUMÉRATION EN C

- L'énumération est un type entier déguisé
- Chaque élément de l'énumération est représenté par un nombre entier
- Pour nous, on utilise une étiquette au lieu du nombre
- Par la suite, on utilise l'énumération comme un type

48

## DÉFINIR ET UTILISER L'ÉNUMÉRATION

- La définition d'une énumération à la forme suivante :
  - Forme générale :

```
enum IdentifiantEnumeration { valeur1, valeur2, ..., valeurN};
```
  - Exemple :

```
enum Couleur {jaune, rouge, bleu, vert};
```
- Affecter une énumération :

```
Couleur c1 = jaune;
```
- Cette action est interdite :

```
jaune = 3;
```

Jaune est une valeur dès quelle est défini dans une énumération

49

## DÉFINIR LA VALEUR DES ÉNUMÉRATIONS

- Comme les énumérations sont des listes d'entier, on peut définir la valeur d'un item
- Pour définir la valeur, on affecte une valeur entière à l'étiquette
- Exemple :

```
enum Couleur {jaune = 5, bleu, vert = 4, bleu = 42};
```

50

## UTILISATION D'UNE ÉNUMÉRATION DANS UN CALCUL ENTIER

- Comme l'énumération est un entier, on peut l'utiliser dans les calculs

```
int p;  
enum Couleur {jaune = 5, bleu, vert = 4, bleu = 42};  
Couleur c = jaune;  
p = 2 + vert * jaune;
```

- La valeur de p est alors 22, car vert = 4 et jaune = 5

**Commencé le** dimanche 19 mars 2023, 15:53**État** Terminé**Terminé le** dimanche 19 mars 2023, 22:54**Temps mis** 7 heures**Note** Pas encore évalué**Question 1**

Correct

Note de 6,00 sur 6,00

Associez les affectations de variable suivantes et leur type C++.

|                      |        |   |
|----------------------|--------|---|
| `Delta = 'r';`       | char   | ✓ |
| Alpha = 5.9f;        | float  | ✓ |
| `Epsilon = "Hello";` | string | ✓ |
| `Zeta = true;`       | bool   | ✓ |
| Beta = 145.;         | double | ✓ |
| `Gamma = 6;`         | int    | ✓ |

Votre réponse est correcte.

La réponse correcte est : `Delta = 'r';` → char, Alpha = 5.9f; → float, `Epsilon = "Hello";` → string, `Zeta = true;` → bool, Beta = 145.; → double, `Gamma = 6;` → int

**Question 2**

Incorrect

Note de 0,00 sur 1,00

À quoi sert l'instruction `#include` ?Réponse : les `#include` contient des déclarations de fonctions, des définitions de constantes, d ✗

La réponse correcte est : Importer une bibliothèque

**Question 3**

Incorrect

Note de 0,00 sur 11,00

À la question précédente, nous avons vu la fonction suivante :

```
Vecteur Rotate(Quaternion rot) {  
    Quaternion p {rot.x, rot.y, rot.z, 0};  
    Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});  
    Quaternion q {norm.x, norm.y, norm.z, rot.w};  
    q = convertToUnityNormQuaternion(q);  
    Quaternion qInverse = inverseQuaternion(q);  
    Quaternion rotatedVector = q * p * qInverse;  
  
    return Vecteur {rotatedVector.x, rotatedVector.y, rotatedVector.z};  
}
```

Donnez le prototype de la fonction suivante : `Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});`

Réponse : `Vecteur Normaliser(Vecteur )`



Les prototypes ont toujours la même forme :

```
<Type de retour> <Nom de la fonction>(<Type du paramètre 1> <Nom du paramètre 1>, <Type du paramètre 2> <Nom du paramètre 2>,  
<...>, <Type du paramètre N> <Nom du paramètre N>);
```

Le prototype doit toujours correspondre à la signature de la fonction terminé d'un point-virgule.

La réponse correcte est : `Vecteur Normaliser(Vecteur );`

## Question 4

Terminé

Noté sur 8,00

Écrivez la fonction, ainsi que le prototype, qui réalise l'algorithme suivant :

```
Procédure EstPremier(n : int)
    Pour i: 2 -> racine de n
        Si n est divisible par i Alors
            Retourner faux
        Retourner vrai
```

Prenez en compte que les bibliothèques ont été correctement incluse.

```
#include <cmath>
```

```
bool EstPremier(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i <= sqrt( ); i++) {
        if (n % i == 0) {
            return false;
        }
    }
}
```

```
bool EstPremier(int n);
```

```
bool EstPair(int n) {
    for (int i = 2; i <= sqrt(n); i++)
        if (n % i == 0)
            return false;
    return true;
}
```

**Question 5**

Terminé

Noté sur 3,00

Écrivez une structure qui représente une personne étudiante. Une personne étudiante est représenté par :

- Un nom
- Un prénom
- Un code permanent

Dans les trois cas, les champs sont des chaînes de caractère. Prenez en compte qu'il y a pas de `using namespace std;` dans le document et les inclusions appropriés ont été faite.

```
struct Etudiant_s {  
    std::string Nom;  
    std::string Prenom;  
    std::string CodePermanent;  
};
```

```
struct PersonneEtudiante {  
    std::string Nom, Prenom, CodePermanent;  
};
```

[◀ Devoir #4](#)

Aller à...

[La machine énigme ►](#)

[Tableau de bord](#) / Mes cours / [Programmation II \(420C21ING1\\_2\)](#) / [Devoirs](#) / Devoir #4

**Commencé le** jeudi 9 mars 2023, 13:28

**État** Terminé

**Terminé le** dimanche 19 mars 2023, 22:53

**Temps mis** 10 jours 8 heures

**Note** Pas encore évalué

## Question 1

Terminé

Noté sur 25,00

En classe, nous avons vu la lecture et l'écriture des fichiers CSV. Pour cette question, nous verrons le **format chevron**. Il s'agit d'un format de stockage inspiré du HTML. Dans ce format, les informations sont stockées par une paire clé/valeur. La clé correspond au nom d'une variable et la valeur correspond au nom de cette variable.

Prenons par exemple la variable suivante `int Exemple = 42;`. Le stockage dans le format chevron sera :

```
<int n="Exemple" v="42">
```

Pour une structure, la structure sera plus complexe. Prenons la structure suivante :

```
struct App {
    string Nom;
    int Version;
};

App aExemple = {"Devoir 4", 1};
```

Le stockage dans le format chevron sera :

```
<App n="aExemple", v=[  
    <string n="Nom" v="Devoir 4">  
    <int n="Version" v="1">]>
```

Pour un tableau, le format chevron sera (pour un tableau d'entier de 3 indices) :

```
<int[] l="3" n="NomDuTableau" v=[  
    <int n="0" v="42">,  
    <int n="1" v="42">,  
    <int n="2" v="42">]>
```

Pour une matrice, le format chevron sera (pour une matrice d'entier de deux par deux cases) :

```
<int[][] l="2" c="2" n="NomDuTableau" v=[  
    <int n="0,0" v="42">,  
    <int n="0,1" v="42">,  
    <int n="1,0" v="42">,  
    <int n="1,1" v="42">]>
```

Créez une fonction qui écrit un tableau de booléen dans le format chevron. Vous ne pouvez pas utiliser `sizeof`. La fonction retournera un string du format.

si:

```
//#include <iostream>
//#include <sstream>
//
//struct App {
//    std::string Nom;
//    int Version;
//};
//App aExemple = { "Devoir 4", 1 };
//
//std::string stockageChevron() {
//    App Aexample;
//    std::string Nom = "Devpor 4";
//    int Version = 1;
```

```
//  
//}  
//  
//int TableauChevron() {  
//    const int l = 3;  
//    int NomDuTableau[l] = {42,42,42};  
//}  
//}  
//  
//int MatriceChevron() {  
//    const int l = 2;  
//    const int c = 2;  
//    int NomDeLaMatrice[l][c] = { {42,42},  
//                                {42,42} };  
//}
```

Alors:

```
#include <iostream>  
#include <sstream>  
  
std::string tableauChevron(const bool* tableau, int taille) {  
    std::ostringstream Chevron;  
  
    Chevron << "< bool [] l = ""4"" n = ""ChevronBool"" v= [ " << std::endl;  
    for (int i = 0; i < taille; ++i) {  
        Chevron << "<bool n = " << i << " v = " << (tableau[i] ? "true" : "false") << ">" << std::endl;  
    }  
    Chevron << "]>" << std::endl;  
  
    return Chevron.str();  
}  
  
int main() {  
    bool monTableau[] = { true, false, true, true, false };  
    std::string resultat = tableauChevron(monTableau, 5);  
  
    std::cout << "Resultat : " << std::endl;  
    std::cout << resultat;  
  
    return 0;  
}  
⚙ test.cpp
```

```
#include <sstream>
#include <fstream>

using std::fstream; using std::ios; using std::string; using std::stringstream;

string EcrireChevron(bool[], size_t, string);

string EcrireChevron(bool tableau[], size_t taille, string NomVariable) {
    stringstream reponse;
    reponse << "<bool[] n="" " << NomVariable << "\n" v=[\n";
    for (size_t i = 0; i < taille; i++)
    {
        reponse << "\t<bool n="" << i << "\n" v=\"";
        reponse << (tableau[i] ? "true" : "false") << "\n";
        if (i < taille - 1)
            reponse << ",\n";
        else
            reponse << "]";
    }
    reponse << ">";
    return reponse.str();
}
```

**Question 2**

Terminé

Noté sur 37,00

Écrivez une fonction qui reçoit un chemin vers un fichier texte et un caractère. La fonction devra lire tout le contenu du fichier et retourner le nombre de répétitions du caractère dans le fichier.

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

using std::fstream; using std::ios;

int compterCaractere(const std::string& cheminFichier, char c) {
    fstream Fichier;

    std::ifstream fichier(cheminFichier);
    if (Fichier.fail()) {
        std::cout << "Impossible d'ouvrir le fichier " << cheminFichier << std::endl;
        exit(EXIT_FAILURE);
    }

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

    int nbRepetitions = 0;
    char caractereCourant;
    while (fichier.get(caractereCourant) || !Fichier.eof()) {
        if (caractereCourant == c) {
            nbRepetitions++;
        }
    }
    return nbRepetitions;
}

int main() {
    std::string cheminFichier = "FicherPourCompter.txt";
    char c = 'e';
    int nbRepetitions = compterCaractere(cheminFichier, c);
    if (nbRepetitions >= 0) {
        std::cout << "Le caractere '" << c << "' apparait " << nbRepetitions << " fois dans le fichier " << cheminFichier << std::endl;
    }
}
```

```
}
```

```
return 0;
```

```
}
```

```
 test.cpp
```

```
int ChercheEtTrouve(string, char);
int ChercheEtTrouve(string chemin, char c) {
    int nbOccurrence = 0;
    fstream fichier;
    fichier.open(chemin, ios::in);
    while (!fichier.eof())
    {
        char caractere;
        caractere = fichier.get();
        nbOccurrence += (caractere == c ? 1 : 0);
    }

    fichier.close();
    return nbOccurrence;
}
```

**Question 3**

Terminé

Noté sur 14,00

Nous avons vu en classe qu'il n'est pas possible d'écrire une variable string dans un fichier binaire. Le type string contient plusieurs éléments en plus des caractères. Nous pouvons utiliser les tableaux de caractères à la place (`char *`). Un autre problème provient de la taille dynamique des chaînes. La solution pourrait être la création d'une structure qui inclurait deux informations : la taille de la chaîne et le tableau de caractère (`char *`).

Écrivez cette structure.

```
struct String {  
    int longueur;  
    char* data;  
};
```

```
struct ChaineBinaire_s {  
    char *str;  
    int Taille;  
};
```



## Question 4

Terminé

Noté sur 46,00

Imaginons que votre programme doit enregistrer la structure suivante :

```
struct Personne_s {
    string Nom, Prenom;
    int Matricule;
    bool EstEtudiant_e;
}
```

À partir de votre structure à la question précédente, écrivez une structure `PersonneFichierBinaire_s` qui remplace les strings de la structure précédente. Par la suite, écrivez une fonction qui écrit, dans un fichier binaire spécifié en paramètre, une instance de la structure `Personne_s`.

Votre fonction doit ajouter l'instance de la `Personne_s` à la fin du fichier. Enfin, la fonction doit retourner un bool pour indiquer que le fichier s'est bien écrit ou non.

Je ne comprend pas vraiment les fichier binaire

```
bool ecrirePersonneDansFichierBinaire(const std::string& nomFichier, const PersonneFichierBinaire_s& personne) {
```

```
    std::fstream fichier(nomFichier, std::ios::out | std::ios::app | std::ios::binary);
```

```
    if (!fichier.eof() || fichier.fail()) {
        std::cerr << "Erreur : impossible d'ouvrir le fichier " << nomFichier << std::endl;
        exit(EXIT_FAILURE);
    }
```

```
// Écriture de la structure PersonneFichierBinaire_s dans le fichier binaire
```

```
    fichier.write((char*)&personne.Nom.longueur, sizeof(personne.Nom.longueur));
    fichier.write(personne.Nom.data, personne.Nom.longueur);
```

```
    fichier.write((char*)&personne.Prenom.longueur, sizeof(personne.Prenom.longueur));
    fichier.write(personne.Prenom.data, personne.Prenom.longueur);
```

```
    fichier.write((char*)&personne.Matricule, sizeof(personne.Matricule));
    fichier.write((char*)&personne.EstEtudiant_e, sizeof(personne.EstEtudiant_e));
```

```
    if (!fichier.eof() || fichier.fail()) {
        std::cerr << "Erreur : impossible d'écrire la structure PersonneFichierBinaire_s dans le fichier " << nomFichier << std::endl;
        fichier.close();
        exit(EXIT_FAILURE);
    }
```

```
// Fermeture du fichier
```

```
fichier.close();

return true;
}
```

 [test.cpp](#)

```
bool EcrirePersonneFichierBinaire(string, Personne_s);
bool EcrirePersonneFichierBinaire(string chemin, Personne_s personne) {
    PersonneFichierBinaire_s pfb{
        {personne.Nom.size(), personne.Nom.c_str()},
        {personne.Prenom.size(), personne.Prenom.c_str()},
        personne.Matricule,
        personne.EstEtudiant_e
    };

    fstream fichier;
    fichier.open(chemin, ios::app | ios::binary);

    fichier.write((char*)&pfb, sizeof(PersonneFichierBinaire_s));

    fichier.close();
    return !fichier.fail();
}
```

**Question 5**

Terminé

Noté sur 41,00

Reprenez les structures de la question précédente et écrivez une fonction qui lit dans un fichier binaire (dont le chemin est en paramètre) une instance de la structure Personne\_s. Dans les paramètres, nous retrouvons la position (index) de l'instance dans le fichier. Enfin, le fichier est écrit avec la fonction de la question précédente.

Ne fonctionne pas, mais je vais essayer tout de même

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

struct String {
    int longueur;
    char* data;
};

struct PersonneFichierBinaire_s {
    String Nom;
    String Prenom;
    int Matricule;
    bool EstEtudiant_e;
};

void lirePersonneDansFichierBinaire(string cheminFichier, PersonneFichierBinaire_s& personne) {
    // Ouvre le fichier binaire en lecture seule
    ifstream fichier(cheminFichier, ios::in | ios::binary);
    if (!fichier.is_open() || fichier.fail()) {
        cout << "Erreur : impossible d'ouvrir le fichier " << cheminFichier << endl;
        exit(EXIT_FAILURE);
    }

    // Lit l'instance de PersonneFichierBinaire_s dans le fichier
    fichier.read(reinterpret_cast<char*>(&personne), sizeof(PersonneFichierBinaire_s));

    // Ferme le fichier
    fichier.close();
}
```

```
void ecrirePersonneDansFichierBinaire(string cheminFichier, const PersonneFichierBinaire_s& personne) {  
  
    ofstream fichier(cheminFichier, ios::out | ios::binary);  
    if (!fichier.is_open() || fichier.fail()) {  
        cout << "Erreur : impossible d'ouvrir le fichier " << cheminFichier << endl;  
        exit(EXIT_FAILURE);  
  
    }  
  
    fichier.write(reinterpret_cast<const char*>(&personne), sizeof(PersonneFichierBinaire_s));  
  
    fichier.close();  
}  
  
  
int main() {  
  
    PersonneFichierBinaire_s personne = { "Doe", "John", 12345, true };  
  
  
    ecrirePersonneDansFichierBinaire("personnes.bin", 0, personne);  
  
  
    PersonneFichierBinaire_s personneLue;  
    lirePersonneDansFichierBinaire("personnes.bin", 0, personneLue);  
  
    cout << "Nom : " << personneLue.Nom.data << endl;  
    cout << "Prenom : " << personneLue.Prenom.data << endl;  
    cout << "Matricule : " << personneLue.Matricule << endl;  
    cout << "EstEtudiant_e : " << personneLue.EstEtudiant_e << endl;  
  
    return 0;  
}
```

 [test.cpp](#)

```
Personne_s LirePersonneFichierBinaire(string, int);
Personne_s LirePersonneFichierBinaire(string chemin, int position) {
    Personne_s personne{};

    int index = -1;
    fstream fichier;
    fichier.open(chemin, ios::app | ios::binary);

    while (!fichier.eof() || index == position)
    {
        int taille{};
        fichier.read((char*)&taille, sizeof(int));
        fichier.read((char*)&personne.Nom, taille);
        fichier.read((char*)&taille, sizeof(int));
        fichier.read((char*)&personne.Prenom, taille);
        fichier.read((char*)&personne.Matricule, sizeof(int));
        fichier.read((char*)&personne.EstEtudiant_e, sizeof(bool));
    }

    fichier.close();
    return personne();
}
```

◀ Devoir #3

Aller à...

Devoir #5 ►

[Tableau de bord](#) / Mes cours / [Programmation II \(420C21ING1\\_2\)](#) / [Devoirs](#) / Devoir #3

**Commencé le** jeudi 2 mars 2023, 14:25

**État** Terminé

**Terminé le** dimanche 5 mars 2023, 23:02

**Temps mis** 3 jours 8 heures

**Note** 15,00 sur 16,00 (93,75%)

Question 1

Correct

Note de 3,00 sur 3,00

Quels sont les trois étapes **du compilateur C++** ?

- a. 1. Préprocesseurs  
2. Édition de lien  
3. Compilation
- b. 1. Préprocesseurs✓  
2. Compilation  
3. Édition de lien
- c. 1. Édition de lien  
2. Préprocesseurs  
3. Compilation

Votre réponse est correcte.

La réponse correcte est :

- 1. Préprocesseurs
- 2. Compilation
- 3. Édition de lien

## Question 2

Correct

Note de 5,00 sur 5,00

Complétez le texte suivant :

Les instructions `#include <fichier>` ✓ permettent d'ajouter le contenu d'un fichier dans un autre. Il est principalement utilisé dans les fichiers **sources** afin d'y *inclure* les déclarations des fonctions, des variables, des structures, etc.

Les instructions `#pragma once` ✓ indique au compilateur d'ajouter un fichier seulement si ce dernier n'a pas déjà été inclus.

Les instructions `#define CONSTANTE valeur` ✓ permettent de définir des valeurs constantes dans le processus de précompilation. Il peut être utilisé pour déclarer des valeurs littérales dans un fichier ou des macros C++.

Les instructions `#ifndef CONSTANTE ... #endif` ✓ indique une section qui doit être compilée seulement si la constante **n'a pas été** définie.

Les instructions `#ifdef CONSTANTE ... #endif` ✓ indique une section qui doit être compilée seulement si la constante **a été** définie.

Votre réponse est correcte.

La réponse correcte est :

Complétez le texte suivant :

Les instructions `[#include <fichier>]` permettent d'ajouter le contenu d'un fichier dans un autre. Il est principalement utilisé dans les fichiers **sources** afin d'y *inclure* les déclarations des fonctions, des variables, des structures, etc.

Les instructions `[#pragma once]` indique au compilateur d'ajouter un fichier seulement si ce dernier n'a pas déjà été inclus.

Les instructions `[#define CONSTANTE valeur]` permettent de définir des valeurs constantes dans le processus de précompilation. Il peut être utilisé pour déclarer des valeurs littérales dans un fichier ou des macros C++.

Les instructions `[#ifndef CONSTANTE ... #endif]` indique une section qui doit être compilée seulement si la constante **n'a pas été** définie.

Les instructions `[#ifdef CONSTANTE ... #endif]` indique une section qui doit être compilée seulement si la constante **a été** définie.

## Question 3

Correct

Note de 1,00 sur 1,00

Vrai ou faux : Le langage cible d'un compilateur C++ est l'assembleur.

Veuillez choisir une réponse.

Vrai ✓

Faux

La réponse correcte est « Vrai ».

**Question 4**

Partiellement correct

Note de 6,00 sur 7,00

**Pourquoi diviser un projet ?**

Il existe plusieurs raisons de diviser un projet. La première raison est de [faciliter] [la lecture], [le développement], [la mise à jour], [la réutilisation du code]. La division permet aussi de simplifier le [partage de code] entre les modules. Par exemple, la diviser permet de réduire les [chargements] inutiles de module dans le code. Finalement, il n'y a aucune limite au nombre de fichiers servant à la division.

Votre réponse est partiellement correcte.

Vous en avez sélectionné correctement 3.

La réponse correcte est :

**Pourquoi diviser un projet ?**

Il existe plusieurs raisons de diviser un projet. La première raison est de [faciliter] [la lecture], [le développement], [la mise à jour], [la réutilisation du code]. La division permet aussi de simplifier le [partage de code] entre les modules. Par exemple, la diviser permet de réduire les [chargements] inutiles de module dans le code. Finalement, il n'y a aucune limite au nombre de fichiers servant à la division.

Commentaire :

◀ Devoir #2

Aller à...

Devoir #4 ►

[Tableau de bord](#) / Mes cours / [Programmation II \(420C21ING1\\_2\)](#) / [Devoirs](#) / Devoir #2

**Commencé le** lundi 13 février 2023, 14:16

**État** Terminé

**Terminé le** dimanche 19 février 2023, 22:40

**Temps mis** 6 jours 8 heures

**Note** **86,00** sur 108,00 (**79,63%**)

**Question 1**

Terminé

Note de 51,00 sur 51,00

En jeu vidéo, pour représenter les transformations géométriques d'un objet de jeu comme un sprite, nous utilisons les objets vectoriels. Grâce aux vecteurs, nous pouvons décrire chaque aspect physique de l'objet : sa position, sa rotation et sa grosseur.

Deux objets vectoriels sont utilisés : les vecteurs (en 3D) et les quaternions. Les premiers (les vecteurs) sont un ensemble de trois nombres décimaux : x, y et z. Les deuxièmes (les quaternions) sont un peu particuliers. Ils sont composés des trois composantes comme les vecteurs, mais nous devons ajouter à ceux-ci le paramètre oméga (représenté par la lettre w).

Enfin, dans un jeu vidéo, tous les objets auront une structure Transform qui réunit la position sous la forme d'un vecteur, la rotation sous la forme d'un quaternion et la grosseur (ou scale) sous la forme d'un vecteur.

Donnez les trois structures demandées.

```
using namespace std;
```

```
struct Vecteur {
```

```
    float x;  
    float y;  
    float z;  
};
```

```
struct Quaternion {
```

```
    float x;  
    float y;  
    float z;  
    float w;  
};
```

```
struct Transform {
```

```
    Vecteur position;  
    Quaternion rotation;  
    Vecteur scale;//grosseur  
};
```

```

struct Vecteur {
    float x, y, z;
};

struct Quaternion {
    float x, y, z, w;
};

struct Transform
{
    Vecteur position, scale;
    Quaternion rotation;
};

```

Commentaire :

Question 2

Correct

Note de 2,00 sur 2,00

Comme nous l'expliquons à la question précédente, les quaternions sont utilisés pour la rotation des objets dans l'espace. Prenons les deux fonctions suivantes :

```

Vecteur Rotate(Quaternion rot) {
    Quaternion p {rot.x, rot.y, rot.z, 0};
    Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});
    Quaternion q {norm.x, norm.y, norm.z, rot.w};
    q = convertToUnityNormQuaternion(q);
    Quaternion qInverse = inverseQuaternion(q);
    Quaternion rotatedVector = q * p * qInverse;

    return Vecteur {rotatedVector.x, rotatedVector.y, rotatedVector.z};
}

Vecteur RotateVect(Vecteur axis, float angle) {
    return Rotate(Quaternion {axis.x, axis.y, axis.z, angle});
}

```

Quels sont les deux prototypes de chaque fonction ?

- a. Vecteur Rotate(Quaternion rot); Vecteur RotateVect(Vecteur axis, float angle); ✓
- b. Vecteur Rotate(rot); Vecteur RotateVect(axis angle);
- c. Vecteur Rotate(rot); Vecteur RotateVect(axis, angle);
- d. Vecteur Rotate(Quaternion rot) Vecteur RotateVect(Vecteur axis, float angle)
- e. Vecteur Rotate(Quaternion rot)/Vecteur RotateVect(Vecteur axis, float angle);

Votre réponse est correcte.

La réponse correcte est : Vecteur Rotate(Quaternion rot); Vecteur RotateVect(Vecteur axis, float angle);

**Question 3**

Partiellement correct

Note de 4,00 sur 11,00

À la question précédente, nous avons vu la fonction suivante :

```
Vecteur Rotate(Quaternion rot) {  
    Quaternion p {rot.x, rot.y, rot.z, 0};  
    Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});  
    Quaternion q {norm.x, norm.y, norm.z, rot.w};  
    q = convertToUnityNormQuaternion(q);  
    Quaternion qInverse = inverseQuaternion(q);  
    Quaternion rotatedVector = q * p * qInverse;  
  
    return Vecteur {rotatedVector.x, rotatedVector.y, rotatedVector.z};  
}
```

Donnez le prototype de la fonction suivante : `Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});`

Réponse : Vecteur Rotate(Quaternion rot)



Les prototypes ont toujours la même forme :

```
<Type de retour> <Nom de la fonction>(<Type du paramètre 1> <Nom du paramètre 1>, <Type du paramètre 2> <Nom du paramètre 2>,  
<...>, <Type du paramètre N> <Nom du paramètre N>);
```

Le prototype doit toujours correspondre à la signature de la fonction terminé d'un point-virgule.

La réponse correcte est : `Vecteur Normaliser(Vecteur );`

Commentaire :

**Question 4**

Partiellement correct

Note de 2,00 sur 11,00

À la question 2, nous avons vu la fonction suivante :

```
Vecteur Rotate(Quaternion rot) {  
    Quaternion p {rot.x, rot.y, rot.z, 0};  
    Vecteur norm = Normaliser(Vecteur {rot.x, rot.y, rot.z});  
    Quaternion q {norm.x, norm.y, norm.z, rot.w};  
    q= convertToUnityNormQuaternion(q);  
    Quaternion qInverse = inverseQuaternion(q);  
    Quaternion rotatedVector = q * p * qInverse;  
  
    return Vecteur {rotatedVector.x, rotatedVector.y, rotatedVector.z};  
}
```

Donnez le prototype de la fonction suivante : `q= convertToUnityNormQuaternion(q);`

Réponse : Vecteur Rotate(Quaternion rot)



Les prototype ont toujours la même forme :

```
<Type de retour> <Nom de la fonction>(<Type du paramètre 1> <Nom du paramètre 1>, <Type du paramètre 2> <Nom du paramètre 2>,  
<...>, <Type du paramètre N> <Nom du paramètre N>);
```

Le prototype doit toujours correspondre à la signature de la fonction terminé d'un point-virgule.

La réponse correcte est : `Quaternion convertToUnityNormQuaternion(Quaternion );`

Commentaire :

## Question 5

Terminé

Note de 27,00 sur 33,00

Comme tous les objets mathématiques, les vecteurs peuvent être additionnés ensemble, multiplier par un nombre réel (scalaire) ou multiplier ensemble (produit scalaire). Créer trois fonctions, ainsi que leurs prototypes, qui réalisent les opérations suivante :

- Addition des vecteurs **a** et **b**
  - Une addition de vecteur correspond à l'addition des composantes **Vecteur c = a + b <=> c.x = a.x + b.x ...**
- Multiplication par un scalaire (nombre réel)
  - Une multiplication par un scalaire correspond à la multiplication des composantes par le scalaire **Scalaire s et Vecteur c = sa <=> c.x = s \* a.x ...**
- Produit scalaire de deux vecteurs
  - Cette opération retourne un scalaire (nombre réel) et elle correspond à la somme des composante des deux vecteurs **Vecteur a, b et Scalaire s = a.x \* b.x + a.y \* ...**

```
#include <iostream>
```

```
using namespace std;
```

```
struct Vecteur {  
    float x;  
    float y;  
    float z;  
};
```

```
struct Quaternion {  
    float x;  
    float y;  
    float z;  
    float w;  
};
```

```
struct Transforme {  
    Vecteur position;  
    Quaternion rotation;  
    Vecteur scale://grosseur  
};
```

```
Vecteur addition(Vecteur a, Vecteur b) {  
    Vecteur position;  
    position.x = a.x + b.x;  
    position.y = a.y + b.y;  
    position.z = a.z + b.z;  
    return position;  
}
```

```

Vecteur scalaire_mult(Vecteur a, float s) {
    Vecteur mul;
    mul.x = a.x * s;
    mul.y = a.y * s;
    mul.z = a.z * s;
    return mul;
}

float scalaire_produit(Vecteur a, Vecteur b) {
    float SP = a.x * b.x + a.y * b.y + a.z * b.z;
    return SP;
}

int main() {
    Vecteur v1 = { 1, 2, 3 };
    Vecteur v2 = { 4, 5, 6 };
    Quaternion q = { 1.5, 1, 1, 1.5 };
    Transforme t = { v1, q, v2 };

    Vecteur v3 = addition(v1, v2);
    Vecteur v4 = scalaire_mult(v1, 2);
    float PS = scalaire_produit(v1, v2);

    cout << "Addition des vecteurs a et b = (" << v3.x << ", " << v3.y << ", " << v3.z << ")" << endl;
    cout << "Multiplication par un scalaire (nombre réel) = (" << v4.x << ", " << v4.y << ", " << v4.z << ")" << endl;
    cout << "Produit scalaire de deux vecteurs = " << PS << endl;
}

```

 [devoir2.cpp](#)

...

```

Vecteur Addition(Vecteur, Vecteur);
Vecteur Addition(Vecteur a, Vecteur b) {
    return Vecteur{ a.x + b.x, a.y + b.y, a.z + b.z };
}

Vecteur Multiplication(float, Vecteur);
Vecteur Multiplication(float s, Vecteur a) {
    return Vecteur{ s * a.x, s * a.y, s * a.z };
}

float ProduitScalaire(Vecteur, Vecteur);
float ProduitScalaire(Vecteur a, Vecteur b) {
    return (a.x * b.x) + (a.y * b.y) + (a.z * b.z);
}

```

...

Commentaire :

◀ Devoir #1

Aller à...

Devoir #3 ►

[Tableau de bord](#) / Mes cours / [Programmation II \(420C21ING1\\_2\)](#) / [Devoirs](#) / Devoir #1

**Commencé le** dimanche 29 janvier 2023, 18:56

**État** Terminé

**Terminé le** dimanche 5 février 2023, 23:59

**Temps mis** 7 jours 5 heures

**Note** **53,00** sur 77,00 (**68,83%**)

## Question 1

Terminé

Note de 15,00 sur 27,00

La table ASCII, incluant la table étendue, contient 256 caractères. À l'aide de boucle, vous devez afficher les caractères 34 à 255 en trois colonnes. Le premier élément de la deuxième colonne est l'élément suivant de la colonne précédente. Exemple, si la première colonne s'arrête au numéro **3**, la deuxième colonne commence au numéro **4**.

Pour chaque caractère, vous devez afficher le caractère en question, son numéro décimal, son équivalent en octet et son équivalent en hexadécimale. Si vous voulez un exemple, regarder à la page 189 du manuel de C11. **Attention !** le caractère et le nombre décimal ont été inversés.

N'oubliez pas d'ajouter un en-tête à vos colonnes. De plus, vous ne pouvez pas utiliser le caractère espace (autre que dans un fill). Tout doit être réalisé à l'aide de boucle et vous devez minimiser le nombre de caractères dans vos strings. Vous devez utiliser des constantes.

N'utilisez pas `cvm23` ou toute autre bibliothèque autre que la bibliothèque standard. Déposez votre fichier source ici.

```
#include <iostream>
#include <iomanip>
#include <conio.h>
#include <string>
```

```
using namespace std;
```

```
int main() {
    const string chara = "charactere";
    const string deci = " decimale ";
    const string hexa = " hexadecimale ";
    const string octa = " octale ";

    for ( int i = 34, j = 74, k = 114; i <= 255; i += 40, j += 40, k += 40) {
        //for (int j = 74; j < 149; j++) {
            cout << setw(2) << chara << setw(13) << deci << octa << hexa << setw(15) << chara << setw(13) << deci << octa << hexa << setw(15) << chara << setw(13) << deci << octa << hexa << endl;
            cout << setw(5) << (char)i << setw(13) << i << oct << setw(10) << i << hex << setw(13) << i << setw(15) << (char)j << setw(12) << j << oct << setw(10) << j << hex << setw(13) << j << setw(13) << (char)k << setw(13) << k << oct << setw(10) << k << hex << setw(10) << k << endl;
        //}
    }
}
```

 [devoir1.cpp](#)

```
#include <iostream>
#include <iomanip>
#include <conio.h>

using namespace std;

int main() {
    const size_t DEBUT = 34, NB_COLONNE = 3, NB_LIGNE = (256 - DEBUT) / NB_COLONNE,
    PREMIERE_COLONNE = 5, MILIEU_COLONNE = 4, DERNIERE_COLONNE = 10;
    for (size_t i = 0; i < 3; i++)
    {
        cout << setw(PREMIERE_COLONNE) << left << "CAR"
            << setw(MILIEU_COLONNE) << "DEC"
            << setw(MILIEU_COLONNE) << "OCT"
            << setw(DERNIERE_COLONNE) << "HEX";
    }

    cout << endl;

    for (size_t i = 0; i < NB_LIGNE; i++)
    {
        for (size_t j = 0; j < NB_COLONNE; j++)
        {
            int c = i + NB_LIGNE * j + DEBUT;
            cout << setw(PREMIERE_COLONNE) << left << (char)c
                << setw(MILIEU_COLONNE) << dec << c
                << setw(MILIEU_COLONNE) << oct << c
                << setw(DERNIERE_COLONNE) << hex << c;
        }

        cout << endl;
    }

    _getch();
}
```

Commentaire :

## Question 2

Terminé

Note de 19,00 sur 22,00

Les couleurs sont généralement représenté grâce à quatre nombre : r, g, b et a. Respectivement, ces quatre lettres représente le niveau de rouge, de vert, de bleu et de transparence dans une couleur. Pour cette question, créer une structure Couleur qui réunit les quatre composantes. **Attention !** vous devez affecter le bon **type** à la structure afin de minimiser l'espace mémoire.

Pour être précis, les trois couleurs (rouge, vert et bleu) sont des nombres limités entre 0 et 255. En règle générale, les couleurs ne dépasseront pas ces valeurs. Il y a une exception pour la couleur alpha. Il s'agit d'un nombre décimale qui va de 0 à 1 où 1 représente une transparence totale et 0 aucune transparence.

Écrivez uniquement la structure dans votre réponse.

```
struct Couleur_s {  
    int r, g, b;  
    float a;  
};  
  
const Couleur COULEUR_ROUGE = { 255, 0, 0, 1.0f };  
const Couleur COULEUR_VERT[4] = { 0, 255, 0, 1.0f };  
const Couleur COULEUR_BLEU[4] = { 0, 0, 255, 1.0f };
```

Je ne suis pas vraiment sûr de comprendre la question

```
struct Couleur  
{  
    uint8_t r, g, b; // unsigned char aurait été une option aussi  
    float a;  
};
```

Commentaire : Il suffit de créer la structure simplement.

**Question 3**

Terminé

Note de 19,00 sur 28,00

Les images sont, dans leur forme la plus simple, une matrice de couleur. Reprenez la structure Couleur de la question #2 et réalisez un programme qui accepte une image (matrice de couleur) de la part de la personne utilisatrice.

Par la suite, le programme devra demander une couleur à retirer. Vous devez, par la suite, retirer cette couleur de l'image. Le retrait de couleur est simple une soustraction du rouge avec le rouge, du vert avec le vert et du bleu avec le bleu. Toutefois, la valeur ne peut pas être négative.

Enfin, ce programme doit afficher l'image (les valeurs des couleurs).

Pour nous simplifier la tâche, créer une image de 3 pixels (pixel = 1 couleur) par 3 pixels. Vous devez demander à la personne utilisatrice d'entrer tous les pixels à la main. Ne faites pas de vérification du flux ou des valeurs entrées.

Tout doit être fait dans une ou des boucles et vous devez utiliser les constantes. N'utilisez pas la bibliothèque CVM. Déposez votre fichier source.

Astuce : Vous devez passer par un int pour lire les nombres.

```
include <iostream>
#include <conio.h>
using namespace std;

int main() {

    struct Couleur_s {
        unsigned int r, v, b, a;
    };

    const Couleur COULEUR_ROUGE[4] = { 255, 0, 0, (float)1.0f };
    const Couleur COULEUR_VERT[4] = { 0, 255, 0, (float)1.0f };
    const Couleur COULEUR_BLEU[4] = { 0, 0, 255, (float)1.0f };

    const int LIG = 3, COL = 3;
    Couleur image[LIG][COL];

    for (int i = 0; i < LIG; i++) {
        for (int j = 0; j < COL; j++) {
            cout << "Entrer la couleur de la façon suivante 0 0 0 0 à la position(" << i << ", " << j << "), puis enter: ";
            cin >> image[i][j].r;
            cin >> image[i][j].v;
            cin >> image[i][j].b;
            cin >> image[i][j].a;
        }
    }
}
```

}

```
Couleur Retirer;
cout << "Retirer la couleur ";
cin >> Retirer.r >> Retirer.v >> Retirer.b >> Retirer.a;

for (int i = 0; i < LIG; i++) {
    for (int j = 0; j < COL; j++) {
        unsigned int newR = image[i][j].r - Retirer.r;
        unsigned int newV = image[i][j].v - Retirer.v;
        unsigned int newB = image[i][j].b - Retirer.b;
        unsigned int newA = (float)image[i][j].a - (float)Retirer.a;
        image[i][j] = { newR, newV, newB, newA };
    }
}

cout << "Resultat:" << endl;
for (int i = 0; i < LIG; i++) {
    for (int j = 0; j < LIG; j++) {
        cout << image[i][j].r << " " << image[i][j].v << " " << image[i][j].b << " " << image[i][j].a << endl;
    }
}

_getch();
}
```

 [test.cpp](#)

```
#include <iostream>
#include <conio.h>

int main() {
    const size_t TAILLE_X = 3, TAILLE_Y = 3;
    Couleur image[TAILLE_X][TAILLE_Y], retrait;

    cout << "Entrez les couleurs de votre image : " << endl;
    for (size_t i = 0; i < TAILLE_X; i++)
    {
        for (size_t j = 0; j < TAILLE_Y; j++)
        {
            int temp = 0;
            cout << "\tPixel [" << i << "][" << j << "]:\n";
            cout << "\t\tRouge = ";
            cin >> temp;
            image[i][j].r = temp;
            cout << "\t\tVert = ";
            cin >> temp;
            image[i][j].g = temp;
            cout << "\t\tBleu = ";
            cin >> temp;
            image[i][j].b = temp;
            cout << "\t\tAlpha = ";
            cin >> image[i][j].a;
            cout << endl;
        }
    }

    int temp = 0;
    cout << "Entrez la couleur à retirer : " << endl;
    cout << "\tRouge = ";
    cin >> temp;
    retrait.r = temp;
    cout << "\tVert = ";
    cin >> temp;
    retrait.g = temp;
    cout << "\tBleu = ";
    cin >> temp;
    retrait.b = temp;
    cout << "\tAlpha = "; cin >> retrait.a;

    cout << "\nRetrait en cours...\n";

    for (size_t i = 0; i < TAILLE_X; i++)
    {
        for (size_t j = 0; j < TAILLE_Y; j++)
        {
            if (image[i][j].r - retrait.r >= 0)
                image[i][j].r -= retrait.r;
            else
                image[i][j].r = 0;

            if (image[i][j].g - retrait.g >= 0)
                image[i][j].g -= retrait.g;
            else
                image[i][j].g = 0;

            if (image[i][j].b - retrait.b >= 0)
                image[i][j].b -= retrait.b;
            else
                image[i][j].b = 0;

            if (image[i][j].a - retrait.a >= 0)
                image[i][j].a -= retrait.a;
            else
                image[i][j].a = 0;
        }
    }
}
```

```
        image[i][j].a = 0;
    }

cout << "La nouvelle image est : \n";
cout << "{\n";
for (size_t i = 0; i < TAILLE_X; i++)
{
    cout << "\t{ ";
    for (size_t j = 0; j < TAILLE_Y; j++)
    {
        cout << "["
            << (int)image[i][j].r << ", "
            << (int)image[i][j].g << ", "
            << (int)image[i][j].b << ", "
            << image[i][j].a << "]";

        if (j < TAILLE_Y - 1)
            cout << ", ";
    }
    cout << "}\n";
}

cout << "}\n";

_getch();
}
```

Commentaire :

◀ Bonus : Conférence sur l'IA

Aller à...

Devoir #2 ►