

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

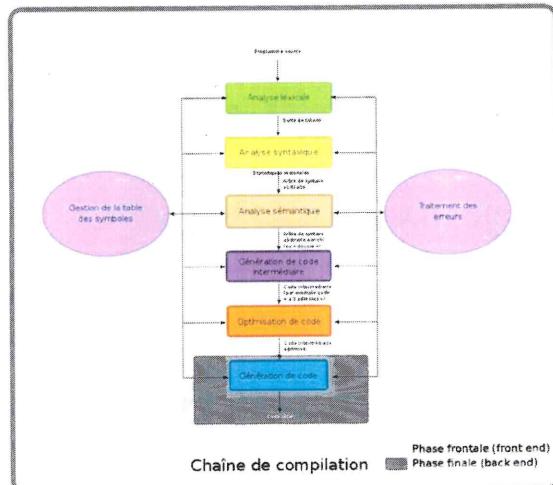
- Traduire un langage source vers un langage cible
- Objectifs :
  - Faciliter l'écriture des instructions
  - Optimiser les opérations
  - Modifier les paradigmes d'écriture
  - Etc.

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++

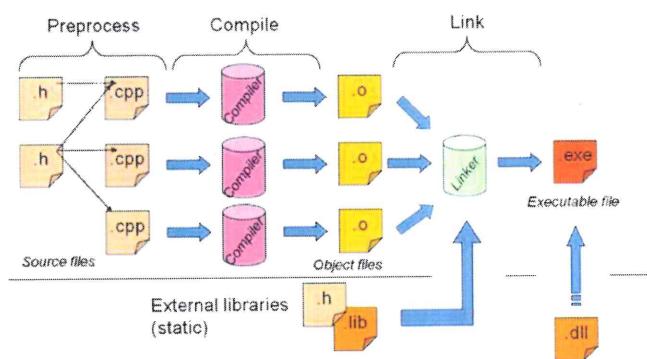


FIGURE 1.1 – Création d'un exécutable lors de la compilation en C++

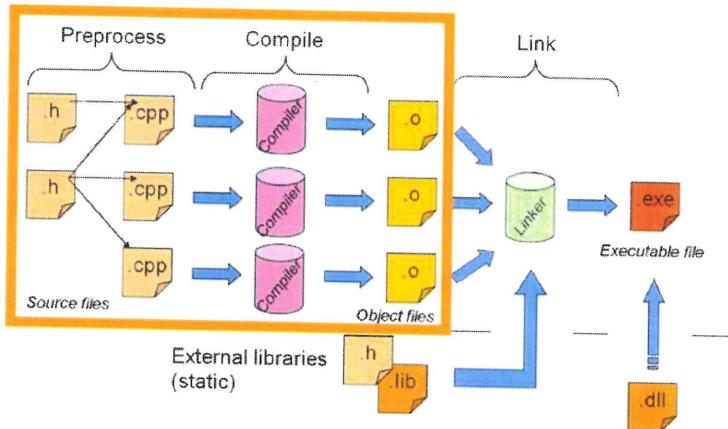
4

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

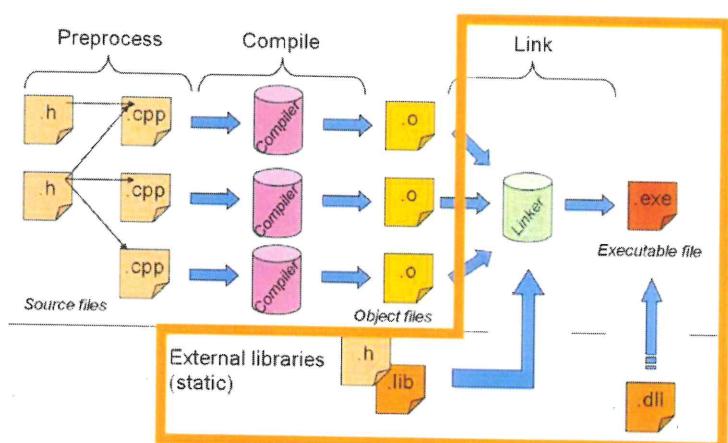
**2. Édition de lien et création de l'exécutable**

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