Modularité

A. Lire un programme 3

B. Compilation séparée 12

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections
- Tests, débogage

Pourquoi aller plus loin ?

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections
- Tests, débogage

Pourquoi aller plus loin ?

Passage à l'échelle !

Résumé des épisodes précédents . . .

Pour le moment nous avons vu les concepts suivants :

- Lecture, écriture
- Instructions conditionnelles et itératives
- Fonctions
- Variables, tableaux, collections
- Tests, débogage

Pourquoi aller plus loin ?

Passage à l'échelle !

Maintenance de « gros » programmes

A. Lire un programme

Dans l'ordre du fichier ?

Exemple : `../Semaine7/sudoku.cpp`

A. Lire un programme

Dans l'ordre du fichier ?

Exemple : `../Semaine7/sudoku.cpp`

Quelles sont les briques de base ? (bottom-up)

Parcourir les fonctions disponibles et leur documentation.

A. Lire un programme

Dans l'ordre du fichier ?

Exemple : `../Semaine7/sudoku.cpp`

Quelles sont les briques de base ? (bottom-up)

Parcourir les fonctions disponibles et leur documentation.

Qui appelle qui ? (top-down)

- Chercher la fonction `main`
- Quelles fonctions appelle-t-elle ?
- Quelles fonctions appellent ces fonctions ?
- Ne rentrer dans les détails que si c'est nécessaire

A. Lire un programme

Dans l'ordre du fichier ?

Exemple : `../Semaine7/sudoku.cpp`

Quelles sont les briques de base ? (bottom-up)

Parcourir les fonctions disponibles et leur documentation.

Qui appelle qui ? (top-down)

- Chercher la fonction `main`
- Quelles fonctions appelle-t-elle ?
- Quelles fonctions appellent ces fonctions ?
- Ne rentrer dans les détails que si c'est nécessaire

Exemple

```
x = cos(theta);
```

A. Lire un programme

Dans l'ordre du fichier ?

Exemple : `../Semaine7/sudoku.cpp`

Quelles sont les briques de base ? (bottom-up)

Parcourir les fonctions disponibles et leur documentation.

Qui appelle qui ? (top-down)

- Chercher la fonction `main`
- Quelles fonctions appelle-t-elle ?
- Quelles fonctions appellent ces fonctions ?
- Ne rentrer dans les détails que si c'est nécessaire

Exemple

```
x = cos(theta);
```

On suppose, a priori, que la fonction `cos` est correcte !

Débogage : prévention

Développement incrémental

- Toujours être « proche de quelque chose qui marche »

Débogage : prévention

Développement incrémental

- Toujours être « proche de quelque chose qui marche »
- Gestion de version (`git`, `mercurial`, ...)

Débogage : prévention

Développement incrémental

- Toujours être « proche de quelque chose qui marche »
- Gestion de version (`git`, `mercurial`, ...)

Spécifications et tests

- Définir précisément la sémantique des fonctions :
  qu'est-ce qu'elles doivent faire
- Tester que la sémantique est respectée sur des exemples

Débogage : prévention

Développement incrémental

- Toujours être « proche de quelque chose qui marche »
- Gestion de version (`git`, `mercurial`, ...)

Spécifications et tests

- Définir précisément la sémantique des fonctions :
  qu'est-ce qu'elles doivent faire
- Tester que la sémantique est respectée sur des exemples

Modularité

- Découpage d'un programme en fonctions : **Cours 4**
- Découpage d'un programme en modules : **Aujourd'hui !**
- Découpage d'un programme en espace de noms : **Plus tard**

B. Compilation séparée

Exemple

Partager une fonction `maxInt` entre plusieurs programmes ?

Déclaration de fonctions

Syntaxe

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Déclaration de fonctions

Syntaxe

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Sémantique

- Le programme **définit** quelque part une fonction `maxInt` avec cette **signature** : type des paramètres et type du résultat

Déclaration de fonctions

Syntaxe

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Sémantique

- Le programme **définit** quelque part une fonction `maxInt` avec cette **signature** : type des paramètres et type du résultat
- Cette définition n'est pas forcément dans le même fichier

Déclaration de fonctions

Syntaxe

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Sémantique

- Le programme **définit** quelque part une fonction `maxInt` avec cette **signature** : type des paramètres et type du résultat
- Cette définition n'est pas forcément dans le même fichier
- Si cette définition n'existe pas ou n'est pas unique, une erreur est déclenchée par le compilateur

Déclaration de fonctions

Syntaxe

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Sémantique

- Le programme **définit** quelque part une fonction `maxInt` avec cette **signature** : type des paramètres et type du résultat
- Cette définition n'est pas forcément dans le même fichier
- Si cette définition n'existe pas ou n'est pas unique, une erreur est déclenchée par le compilateur
- Cette erreur est déclenchée au moment où l'on combine les différents fichiers (édition de liens ; voir plus loin)

Compilation séparée

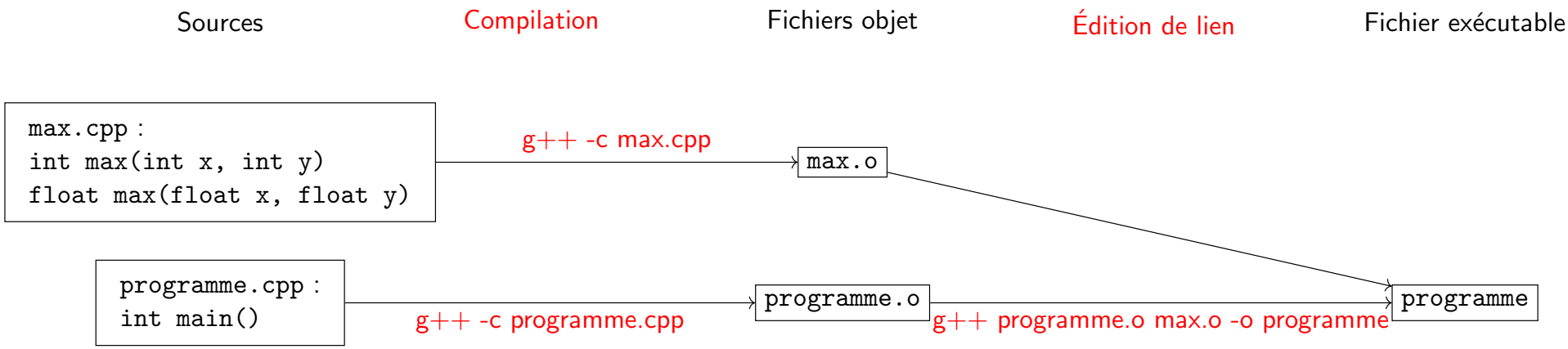
- Un programme peut être composé de plusieurs **fichiers source**
  Contenu :
  - Des **définitions** de fonctions
  - Des variables globales, ...

Compilation séparée

- Un programme peut être composé de plusieurs **fichiers source**
  Contenu :
  - Des **définitions** de fonctions
  - Des variables globales, ...
- Chaque fichier source est compilé en un **fichier objet** (extension : .o)
  Contenu :
  - Le code binaire des fonctions, ...

Compilation séparée

- Un programme peut être composé de plusieurs **fichiers source**
  Contenu :
  - Des **définitions** de fonctions
  - Des variables globales, ...
- Chaque fichier source est compilé en un **fichier objet** (extension : .o)
  Contenu :
  - Le code binaire des fonctions, ...
- L'**éditeur de lien** combine plusieurs fichiers objet en un **fichier exécutable**



Compilation séparée (2)

Au moment de l'édition de lien

- Chaque fonction utilisée doit être définie une et une seule fois
- La fonction `main` doit être définie une et une seule fois

Compilation séparée (2)

Au moment de l'édition de lien

- Chaque fonction utilisée doit être définie une et une seule fois
- La fonction `main` doit être définie une et une seule fois

Quelques variantes autour des fichiers objets

- Bibliothèques (.a) :
  Une archive contenant plusieurs fichiers objets .o
- Bibliothèques dynamiques (.so) :
  Édition de lien dynamique au lancement du programme

Fichiers d'entête

Fichier .h contenant la **déclaration** des fonctions **définies** dans le fichier .cpp correspondant

Exemple (Fichier d'entête `max.h`)

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Fichiers d'entête

Fichier .h contenant la **déclaration** des fonctions **définies** dans le fichier .cpp correspondant

Exemple (Fichier d'entête `max.h`)

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Syntaxe (Utilisation d'un fichier d'entête)

```
#include "max.h"
```

Fichiers d'entête

Fichier .h contenant la **déclaration** des fonctions **définies** dans le fichier .cpp correspondant

Exemple (Fichier d'entête `max.h`)

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Syntaxe (Utilisation d'un fichier d'entête)

```
#include "max.h"
```

Sémantique

Utiliser la bibliothèque `max`

Fichiers d'entête

Fichier .h contenant la **déclaration** des fonctions **définies** dans le fichier .cpp correspondant

Exemple (Fichier d'entête `max.h`)

compilation-separee/max.h

```
int maxInt(int a, int b);
```

Syntaxe (Utilisation d'un fichier d'entête)

```
#include "max.h"
```

Sémantique

Utiliser la bibliothèque `max`

Implantation en C++

- Équivalent à copier-coller le contenu de `max.h` à l'emplacement du `#include "max.h"`
- Géré par le préprocesseur (cpp)

Inclusion de fichiers d'entêtes standards

Syntaxe

```
#include <iostream>
```

Inclusion de fichiers d'entêtes standards

Syntaxe

```
#include <iostream>
```

Sémantique

- Charge la déclaration de toutes les fonctions définies dans la bibliothèque standard `iostream` de C++

Inclusion de fichiers d'entêtes standards

Syntaxe

```
#include <iostream>
```

Sémantique

- Charge la déclaration de toutes les fonctions définies dans la bibliothèque standard `iostream` de C++
- Le fichier `iostream` est recherché dans les répertoires standards du système

Inclusion de fichiers d'entêtes standards

Syntaxe

```
#include <iostream>
```

Sémantique

- Charge la déclaration de toutes les fonctions définies dans la bibliothèque standard `iostream` de C++
- Le fichier `iostream` est recherché dans les répertoires standards du système
- Sous linux : `/usr/include`, ...

Résumé : implantation d'une bibliothèque en C++

Écrire un fichier d'entête (max.h)

- La déclaration de toutes les fonctions publiques
- **Avec leur documentation !**

Résumé : implantation d'une bibliothèque en C++

Écrire un fichier d'entête (max.h)

- La déclaration de toutes les fonctions publiques
- **Avec leur documentation !**

Écrire un fichier source (max.cpp)

- La définition de toutes les fonctions
- **Inclure le fichier .h !**

Résumé : implantation d'une bibliothèque en C++

Écrire un fichier d'entête (max.h)

- La déclaration de toutes les fonctions publiques
- **Avec leur documentation !**

Écrire un fichier source (max.cpp)

- La définition de toutes les fonctions
- **Inclure le fichier .h !**

Écrire un fichier de tests (maxTest.cpp)

- Les fonctions de tests
- Une fonction `main` lançant tous les tests

Résumé : implantation d'une bibliothèque en C++

Écrire un fichier d'entête (max.h)

- La déclaration de toutes les fonctions publiques
- **Avec leur documentation !**

Écrire un fichier source (max.cpp)

- La définition de toutes les fonctions
- **Inclure le fichier .h !**

Écrire un fichier de tests (maxTest.cpp)

- Les fonctions de tests
- Une fonction `main` lançant tous les tests
- Avec `Code::Blocks` : créer un projet contenant `max.h, max.cpp, maxTest.cpp`

Résumé : utilisation d'une bibliothèque en C++

Inclusion des entêtes

```
#include <iostream>  // fichier d'entête standard
#include "max.h"     // fichier d'entête perso
```

Résumé : utilisation d'une bibliothèque en C++

Inclusion des entêtes

```
#include <iostream>  // fichier d'entête standard
#include "max.h"     // fichier d'entête perso
```

Raccourci pour les espaces de nom (optionnel)

```
using namespace std;
```

Résumé : utilisation d'une bibliothèque en C++

Inclusion des entêtes

```
#include <iostream>  // fichier d'entête standard
#include "max.h"     // fichier d'entête perso
```

Raccourci pour les espaces de nom (optionnel)

```
using namespace std;
```

Compilation

- Donner la bibliothèque à charger à l'éditeur de lien

Résumé : utilisation d'une bibliothèque en C++

Inclusion des entêtes

```
#include <iostream>  // fichier d'entête standard
#include "max.h"     // fichier d'entête perso
```

Raccourci pour les espaces de nom (optionnel)

```
using namespace std;
```

Compilation

- Donner la bibliothèque à charger à l'éditeur de lien
- Avec `Code::Blocks` :
  - Créer un projet pour chaque fichier exécutable (programme)
  - Y mettre tous les fichiers qui doivent être compilés ensemble :
    - Fichiers sources : .cpp
    - Fichiers d'entêtes : .h

Digression : surcharge de fonctions ♣

Exemple (la fonction `monMax` : `Exemples/max-surcharge.cpp`)

- Pour les entiers, les réels, les chaînes de caractères, ...

max-surcharge.cpp

```
int monMax(int a, int b) {
```

max-surcharge.cpp

```
string monMax(string a, string b) {
```

- À deux ou trois arguments (et plus ?)

max-surcharge.cpp

```
int monMax(int a, int b, int c) {
```

Digression : surcharge de fonctions ♣

Exemple (la fonction `monMax` : `Exemples/max-surcharge.cpp`)

- Pour les entiers, les réels, les chaînes de caractères, ...

max-surcharge.cpp

```
int monMax(int a, int b) {
```

max-surcharge.cpp

```
string monMax(string a, string b) {
```

- À deux ou trois arguments (et plus ?)

max-surcharge.cpp

```
int monMax(int a, int b, int c) {
```

Surcharge

- En C++, on peut avoir plusieurs fonctions avec le même nom et des signatures différentes
- Idem en Java, mais pas en C ou en Python par exemple !
- Il est recommandé que toutes les fonctions ayant le même nom aient la même **sémantique**

Digression : templates ♣

- L'exemple précédent n'est pas satisfaisant (duplication)
- Correctif : les **templates** pour écrire du code **générique**
  La fonction `monMax` suivante est valide pour tout type sur lequel on peut faire des comparaisons :

max-surcharge-templates.cpp

```
template<class T>
T monMax(T a, T b) {
    if ( a >= b ) {
        return a;
    } else {
        return b;
    }
}
```

Digression : templates ♣

- L'exemple précédent n'est pas satisfaisant (duplication)
- Correctif : les **templates** pour écrire du code **générique**
  La fonction `monMax` suivante est valide pour tout type sur lequel on peut faire des comparaisons :

max-surcharge-templates.cpp

```
template<class T>
T monMax(T a, T b) {
    if ( a >= b ) {
        return a;
    } else {
        return b;
    }
}
```

- Ce sera pour un autre cours !

Digression : espaces de noms ♣

Problème

Conflit entre bibliothèques définissant des fonctions avec le même nom et la même signature !

Digression : espaces de noms ♣

Problème

Conflit entre bibliothèques définissant des fonctions avec le même nom et la même signature !

Solution

Isoler chaque bibliothèque dans un **espace de noms**

Digression : espaces de noms ♣

Problème

Conflit entre bibliothèques définissant des fonctions avec le même nom et la même signature !

Solution

Isoler chaque bibliothèque dans un **espace de noms**

Exemple

La bibliothèque standard C++ utilise l'espace de nom `std` :

bonjour-std.cpp

```
#include <iostream>

int main() {
    std::cout << "Bonjour !" << std::endl;
    return 0;
}
```

Digression : espaces de noms ♣

Problème

Conflit entre bibliothèques définissant des fonctions avec le même nom et la même signature !

Solution

Isoler chaque bibliothèque dans un **espace de noms**

Exemple

La bibliothèque standard C++ utilise l'espace de nom `std` :

bonjour-std.cpp

```
#include <iostream>

int main() {
    std::cout << "Bonjour !" << std::endl;
    return 0;
}
```

Vous verrez plus tard comment créer vos propres espaces de noms

Résumé de la séance

Quelques indications sur comment lire un programme

Résumé de la séance

Quelques indications sur comment lire un programme

Compilation séparée pour la modularité

- Découper un programme non seulement en fonctions, mais en fichiers

Résumé de la séance

Quelques indications sur comment lire un programme

Compilation séparée pour la modularité

- Découper un programme non seulement en fonctions, mais en fichiers
- Bibliothèque de fonctions réutilisables entre programmes

Résumé de la séance

Quelques indications sur comment lire un programme

Compilation séparée pour la modularité

- Découper un programme non seulement en fonctions, mais en fichiers
- Bibliothèque de fonctions réutilisables entre programmes
  - fichier d'entêtes : `max.h`
  - fichier source : `max.cpp`
  - fichier de tests : `maxTest.cpp`

Résumé de la séance

Quelques indications sur comment lire un programme

Compilation séparée pour la modularité

- Découper un programme non seulement en fonctions, mais en fichiers
- Bibliothèque de fonctions réutilisables entre programmes
  - fichier d'entêtes : `max.h`
  - fichier source : `max.cpp`
  - fichier de tests : `maxTest.cpp`

Digressions

- Surcharge
- Templates
- Espaces de noms