





# Modularité, compilation séparée

A. Digression : programmes et compilation .....	4
B. Compilation séparée .....	55
C. Digressions : surcharge, templates, espaces de noms, ... ..	85

## Comment vous sentez-vous en ce début de cours ?

	Curieuse, heureux, joyeux, ...
	Grumble, grumble, ...
	Inquiète
	Gros baillement

## Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections
- ▶ Entrées-sorties, fichiers
- ▶ Complexité

Pourquoi aller plus loin ?

## Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections
- ▶ Entrées-sorties, fichiers
- ▶ Complexité

Pourquoi aller plus loin ?

Passage à l'échelle !

## Résumé des épisodes précédents ...

Pour le moment nous avons vu les concepts suivants :

- ▶ Instructions conditionnelles et itératives
- ▶ Fonctions (avec documentation et tests)
- ▶ Variables, tableaux (2D), collections
- ▶ Entrées-sorties, fichiers
- ▶ Complexité

Pourquoi aller plus loin ?

Passage à l'échelle !

Maintenance de « gros » programmes

## A. Digression : programmes et compilation

Retour sur la recette de la mousse au chocolat

### Ingrédients

250g de chocolat, 125g de beurre, 6 œufs, 50 g de sucre, café

### Étapes

- ▶ Faire fondre le chocolat avec deux cuillères d'eau
- ▶ Ajouter le beurre, laisser refroidir puis ajouter les jaunes
- ▶ Ajouter le sucre et comme parfum un peu de café
- ▶ Battre les blancs jusqu'à former une neige uniforme
- ▶ Ajouter au mélange.

## A. Digression : programmes et compilation

Retour sur la recette de la mousse au chocolat

### Ingrédients

250g de chocolat, 125g de beurre, 6 œufs, 50 g de sucre, café

### Étapes

- ▶ Faire fondre le chocolat avec deux cuillères d'eau
- ▶ Ajouter le beurre, laisser refroidir puis ajouter les jaunes
- ▶ Ajouter le sucre et comme parfum un peu de café
- ▶ Battre les blancs jusqu'à former une neige uniforme
- ▶ Ajouter au mélange.

Est-ce bien un programme ?

## A. Digression : programmes et compilation

Retour sur la recette de la mousse au chocolat

### Ingrédients

250g de chocolat, 125g de beurre, 6 œufs, 50 g de sucre, café

### Étapes

- ▶ Faire fondre le chocolat avec deux cuillères d'eau
- ▶ Ajouter le beurre, laisser refroidir puis ajouter les jaunes
- ▶ Ajouter le sucre et comme parfum un peu de café
- ▶ Battre les blancs jusqu'à former une neige uniforme
- ▶ Ajouter au mélange.

Est-ce bien un programme ?

*Programme* (rappel) : séquence d'*instructions* qui spécifie, étape par étape, les opérations à effectuer pour obtenir, à partir des *entrées*, un résultat, *la sortie*.



## A. Digression : programmes et compilation

C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »

## A. Digression : programmes et compilation

C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »

Trop détaillé, illisible, non portable

- ▶ « Préparer une mousse au chocolat »

Trop abstrait et non informatif

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »  
Trop abstrait et non informatif
- ▶ « avec deux cuillères d'eau »

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »  
Trop abstrait et non informatif
- ▶ « avec deux cuillères d'eau »  
Ambigu : c'est combien une cuillère ?

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »  
Trop abstrait et non informatif
- ▶ « avec deux cuillères d'eau »  
Ambigu : c'est combien une cuillère ?
- ▶ « Zwei Eiweiß in Eischnee schlagen »

## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »  
Trop abstrait et non informatif
- ▶ « avec deux cuillères d'eau »  
Ambigu : c'est combien une cuillère ?
- ▶ « Zwei Eiweiß in Eischnee schlagen »  
Dans quelle langue ?



## A. Digression : programmes et compilation

### C'est quoi une instruction ?

- ▶ « Lever le bras de 10 cm, tendre la main vers la gauche, prendre la casserole rouge, ... »  
Trop détaillé, illisible, non portable
- ▶ « Préparer une mousse au chocolat »  
Trop abstrait et non informatif
- ▶ « avec deux cuillères d'eau »  
Ambigu : c'est combien une cuillère ?
- ▶ « Zwei Eiweiß in Eischnee schlagen »  
Dans quelle langue ?

Quelles sont les instructions compréhensibles par un ordinateur ?

# Assembleur

## Exercice : exécuter ce fragment d'**assembleur**

[puissance-quatre-extrait.s](#)

```
mov    -0x1c(%rbp), %edx
mov    -0x1c(%rbp), %eax
imul   %edx, %eax
mov    %eax, -0x18(%rbp)
mov    -0x18(%rbp), %eax
imul   -0x18(%rbp), %eax
mov    %eax, -0x14(%rbp)
```

## Indications

- ▶ %eax, %edx : deux *registres* (cases mémoire du processeur)
- ▶ -0x14(%rbp), ..., -0x1c(%rbp) : autres cases mémoire
- ▶ Initialiser le contenu de la case %-0x1c(%rbp) à 3
- ▶ mov a, b : copier le contenu de la case a dans la case b
- ▶ imul a, b : multiplier le contenu de a par celui de b et mettre le résultat dans b

## Cycle de vie d'un programme : motivation

Ce que je veux :

« Calculer la puissance 4<sup>e</sup> d'un nombre »

## Cycle de vie d'un programme : motivation

Ce que je veux :

« Calculer la puissance 4<sup>e</sup> d'un nombre »

Ce que l'ordinateur sait faire :

[puissance-quatre-extrait.s](#)

```
...  
mov    -0x1c(%rbp), %edx  
mov    -0x1c(%rbp), %eax  
imul   %edx, %eax  
mov    %eax, -0x18(%rbp)  
mov    -0x18(%rbp), %eax  
imul   -0x18(%rbp), %eax  
mov    %eax, -0x14(%rbp)  
...
```

## Cycle de vie d'un programme : motivation

Ce que je veux :

« Calculer la puissance 4<sup>e</sup> d'un nombre »

Ce que l'ordinateur sait faire :

[puissance-quatre-extrait.s](#)

```
...  
mov    -0x1c(%rbp), %edx  
mov    -0x1c(%rbp), %eax  
imul  %edx, %eax  
mov    %eax, -0x18(%rbp)  
mov    -0x18(%rbp), %eax  
imul  -0x18(%rbp), %eax  
mov    %eax, -0x14(%rbp)  
...
```

Cela ne va pas se faire tout seul ...

# Cycle de vie d'un programme : formalisation et algorithme

## Problème

« Calculer la puissance 4<sup>e</sup> d'un nombre »

# Cycle de vie d'un programme : formalisation et algorithme

## Problème

« Calculer la puissance 4<sup>e</sup> d'un nombre »

## Formalisation

Spécification des *entrées* et des *sorties*

Scénario d'utilisation : « l'utilisateur rentre au clavier un nombre entier  $x$  ;  
l'ordinateur affiche en retour la valeur de  $x^4$  à l'écran »

# Cycle de vie d'un programme : formalisation et algorithme

## Problème

« Calculer la puissance 4<sup>e</sup> d'un nombre »

## Formalisation

Spécification des *entrées* et des *sorties*

Scénario d'utilisation : « l'utilisateur rentre au clavier un nombre entier  $x$  ; l'ordinateur affiche en retour la valeur de  $x^4$  à l'écran »

## Recherche d'un algorithme

Comment on résout le problème ?

Quel *traitement* appliquer à l'entrée pour obtenir la sortie désirée ?



# Cycle de vie d'un programme : formalisation et algorithme

## Problème

« Calculer la puissance 4<sup>e</sup> d'un nombre »

## Formalisation

Spécification des *entrées* et des *sorties*

Scénario d'utilisation : « l'utilisateur rentre au clavier un nombre entier  $x$  ; l'ordinateur affiche en retour la valeur de  $x^4$  à l'écran »

## Recherche d'un algorithme

Comment on résout le problème ?

Quel *traitement* appliquer à l'entrée pour obtenir la sortie désirée ?

On note que  $x^4 = x * x * x * x = (x^2)^2$

# Cycle de vie d'un programme : formalisation et algorithme

## Problème

« Calculer la puissance 4<sup>e</sup> d'un nombre »

## Formalisation

Spécification des *entrées* et des *sorties*

Scénario d'utilisation : « l'utilisateur rentre au clavier un nombre entier  $x$  ; l'ordinateur affiche en retour la valeur de  $x^4$  à l'écran »

## Recherche d'un algorithme

Comment on résout le problème ?

Quel *traitement* appliquer à l'entrée pour obtenir la sortie désirée ?

On note que  $x^4 = x * x * x * x = (x^2)^2$

## Algorithme

- ▶ calculer  $x * x$
- ▶ prendre le résultat et faire de même

# La notion d'algorithme

## Définition (Algorithme)

- ▶ Description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites

# La notion d'algorithme

## Définition (Algorithme)

- ▶ Description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites
- ▶ Succession finie et non ambiguë d'opérations clairement posées

# La notion d'algorithme

## Définition (Algorithme)

- ▶ Description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites
- ▶ Succession finie et non ambiguë d'opérations clairement posées

## Notes

- ▶ Doit donc toujours se terminer !

# La notion d'algorithme

## Définition (Algorithme)

- ▶ Description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites
- ▶ Succession finie et non ambiguë d'opérations clairement posées

## Notes

- ▶ Doit donc toujours se terminer !
- ▶ Conçu pour communiquer entre humains

# La notion d'algorithme

## Définition (Algorithme)

- ▶ Description formelle d'un procédé de traitement qui permet, à partir d'un ensemble d'informations initiales, d'obtenir des informations déduites
- ▶ Succession finie et non ambiguë d'opérations clairement posées

## Notes

- ▶ Doit donc toujours se terminer !
- ▶ Conçu pour communiquer entre humains
- ▶ Concept indépendant du langage dans lequel il est écrit

# Cycle de vie d'un programme : implantation

Et maintenant ?

L'algorithme s'adresse à un humain

On veut l'expliquer à un ordinateur ...



# Cycle de vie d'un programme : implantation

Et maintenant ?

L'algorithme s'adresse à un humain

On veut l'expliquer à un ordinateur ...

qui est stupide ; et ne comprend pas le français !

# Cycle de vie d'un programme : implantation

## Et maintenant ?

L'algorithme s'adresse à un humain

On veut l'expliquer à un ordinateur ...

qui est stupide ; et ne comprend pas le français !

## Écriture d'un programme

# Cycle de vie d'un programme : implantation

## Et maintenant ?

L'algorithme s'adresse à un humain

On veut l'expliquer à un ordinateur ...

qui est stupide ; et ne comprend pas le français !

## Écriture d'un programme

En assembleur ???

# Cycle de vie d'un programme : implantation

## Et maintenant ?

L'algorithme s'adresse à un humain

On veut l'expliquer à un ordinateur ...

qui est stupide ; et ne comprend pas le français !

## Écriture d'un programme

En assembleur ???

- ▶ Trop détaillé
- ▶ Non portable
- ▶ Illisible pour l'humain !

## Cycle de vie d'un programme (4)

puissance-quatre.cpp

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

int main() {
    int x, xCarre, xPuissanceQuatre;

    cout << "Entrez un entier: ";
    cin >> x;

    xCarre = x * x;
    xPuissanceQuatre = xCarre * xCarre;

    cout << "La puissance quatrième de " << x
         << " est " << xPuissanceQuatre << endl;

    return 0;
}
```

# Niveaux de langages de programmation

Langage machine (binaire)

Un programme y est directement compréhensible par la machine

# Niveaux de langages de programmation

## Langage machine (binaire)

Un programme y est directement compréhensible par la machine

## Langage d'assemblage (ou *assembleur*)

Un programme y est directement traduisible en langage machine

# Niveaux de langages de programmation

## Langage machine (binaire)

Un programme y est directement compréhensible par la machine

## Langage d'assemblage (ou *assembleur*)

Un programme y est directement traduisible en langage machine

## Langage de programmation

Un programme doit être *transformé* pour être compris par la machine



# Niveaux de langages de programmation

## Langage machine (binaire)

Un programme y est directement compréhensible par la machine

## Langage d'assemblage (ou *assembleur*)

Un programme y est directement traduisible en langage machine

## Langage de programmation

Un programme doit être *transformé* pour être compris par la machine

## Question

Comment faire cette transformation ? À la main ?

# Transformer en binaire : les interpréteurs

## Exemple : interpréteur C++ dans Jupyter

puissance-quatre.ipynb

```
In [1]: int x;
```

```
In [2]: x = 5
```

```
Out[2]: 5  
type: int
```

```
In [3]: int xCarre = x * x;  
int xPuissance4 = xCarre * xCarre;
```

```
In [4]: xPuissance4
```

```
Out[4]: 625  
type: int
```

# Transformer en binaire : les interpréteurs

## Exemple : interpréteur C++ dans Jupyter

puissance-quatre.ipynb

```
In [1]: int x;
```

```
In [2]: x = 5
```

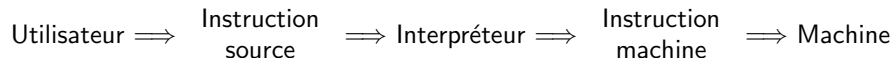
```
Out[2]: 5  
type: int
```

```
In [3]: int xCarre = x * x;  
int xPuissance4 = xCarre * xCarre;
```

```
In [4]: xPuissance4
```

```
Out[4]: 625  
type: int
```

## Chaîne de production



# Transformer en binaire : les interpréteurs

## Exemple : interpréteur C++ dans Jupyter

puissance-quatre.ipynb

```
In [1]: int x;
```

```
In [2]: x = 5
```

```
Out[2]: 5  
type: int
```

```
In [3]: int xCarre = x * x;  
int xPuissance4 = xCarre * xCarre;
```

```
In [4]: xPuissance4
```

```
Out[4]: 625  
type: int
```

## Chaîne de production

Utilisateur  $\implies$  Instruction source  $\implies$  Interpréteur  $\implies$  Instruction machine  $\implies$  Machine

## Exemples (quelques langages interprétés)

Basic, LISP, Perl, Python, C++, ...

# Transformer en binaire : les compilateurs

Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`

# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`

# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`

# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`
- ▶ Exécution : `./puissance-quatre`



# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`
- ▶ Exécution : `./puissance-quatre`
- ▶ Fabrication de l'assembleur : `g++ -S puissance-quatre.cpp`

# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

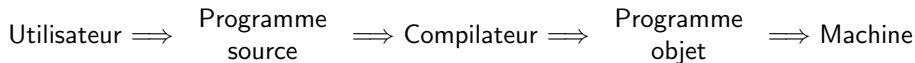
- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`
- ▶ Exécution : `./puissance-quatre`
- ▶ Fabrication de l'assembleur : `g++ -S puissance-quatre.cpp`
- ▶ Programme en assembleur : `puissance-quatre.s`

# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`
- ▶ Exécution : `./puissance-quatre`
- ▶ Fabrication de l'assembleur : `g++ -S puissance-quatre.cpp`
- ▶ Programme en assembleur : `puissance-quatre.s`

## Chaîne de production

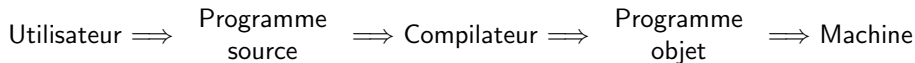


# Transformer en binaire : les compilateurs

## Exemple : compilation en C++

- ▶ Programme source : `puissance-quatre.cpp`
- ▶ Compilation :  
`g++ puissance-quatre.cpp -o puissance-quatre`
- ▶ Programme objet (ou binaire) : `puissance-quatre`
- ▶ Exécution : `./puissance-quatre`
- ▶ Fabrication de l'assembleur : `g++ -S puissance-quatre.cpp`
- ▶ Programme en assembleur : `puissance-quatre.s`

## Chaîne de production



## Exemples (Exemples de langages compilés)

Pascal, C, C++, ADA, FORTRAN, Java, ...

# Cycle de vie d'un programme : exécution, mise au point

## Exécuter le programme

- ▶ Autant de fois que l'on veut !

# Cycle de vie d'un programme : exécution, mise au point

## Exécuter le programme

- ▶ Autant de fois que l'on veut !

## Tester que le programme fonctionne

- ▶ Cas particuliers!!!

# Cycle de vie d'un programme : exécution, mise au point

## Exécuter le programme

- ▶ Autant de fois que l'on veut !

## Tester que le programme fonctionne

- ▶ Cas particuliers!!!

## Améliorer le programme

- ▶ Correction d'erreurs
- ▶ Optimisation du programme (rapidité, consommation mémoire)
- ▶ Optimisation de l'algorithme
- ▶ Amélioration du programme (lisibilité, généralisation)

# Cycle de vie d'un programme : résumé

## Méthodologie

1. Énoncé du problème
2. Formalisation (quel est le problème précisément)
3. Recherche d'un algorithme (comment résoudre le problème?)
4. Programmation (implantation)
5. Interprétation / Compilation
6. Exécution
7. Mise au point (test, débogage, optimisation, diffusion)



## B. Compilation séparée

compilation-separee-avant/programme1.cpp

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

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}

int main() {
    cout << monMax(1, 3) << endl;
    return 0;
}
```

compilation-separee-avant/programme2.cpp

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

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}

int main() {
    cout << "Entrez a et b:" << endl;
    int a, b;
    cin >> a >> b;
    cout << "Le maximum est: "
         << monMax(a, b) << endl;
    return 0;
}
```

## B. Compilation séparée

compilation-separee-avant/programme1.cpp

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

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}

int main() {
    cout << monMax(1, 3) << endl;
    return 0;
}
```

compilation-separee-avant/programme2.cpp

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

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}

int main() {
    cout << "Entrez a et b:" << endl;
    int a, b;
    cin >> a >> b;
    cout << "Le maximum est: "
         << monMax(a, b) << endl;
    return 0;
}
```

### Problème

Partager une fonction `monMax` entre ces deux programmes ?

## Exemple : la bibliothèque max

compilation-separee/max.h

```
/** La fonction max
 * @param x, y deux entiers
 * @return un entier,
 * le maximum de x et de y
 */
int monMax(int a, int b);
```

# Exemple : la bibliothèque max

compilation-separee/max.h

```
/** La fonction max
 * @param x, y deux entiers
 * @return un entier,
 * le maximum de x et de y
 */
int monMax(int a, int b);
```

compilation-separee/max.cpp

```
#include "max.h"

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}
```

## Exemple : la bibliothèque max

compilation-separee/max.h

```
/** La fonction max
 * @param x, y deux entiers
 * @return un entier,
 * le maximum de x et de y
 */
int monMax(int a, int b);
```

compilation-separee/max.cpp

```
#include "max.h"

int monMax(int a, int b) {
    if ( a >= b )
        return a;
    else
        return b;
}
```

## Deux programmes utilisant cette bibliothèque

compilation-separee/programme1.cpp

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

int main() {
    cout << monMax(1, 3) << endl;
    return 0;
}
```

compilation-separee/programme2.cpp

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

int main() {
    cout << "Entrez a et b:" << endl;
    int a, b;
    cin >> a >> b;
    cout << "Le maximum est: "
         << monMax(a, b) << endl;
    return 0;
}
```

## Exemple : Les tests de la bibliothèque max

compilation-separee/maxTest.cpp

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

#include "max.h"

/** Infrastructure minimale de test */
#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ <<

void monMaxTest() {
    ASSERT( monMax(2,3) == 3 );
    ASSERT( monMax(5,2) == 5 );
    ASSERT( monMax(1,1) == 1 );
}

int main() {
    monMaxTest();
}
```

# Déclaration de fonctions

## Syntaxe

`compilation-separee/max.h`

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

# Déclaration de fonctions

## Syntaxe

`compilation-separee/max.h`

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

## Sémantique

- ▶ Le programme **défini** quelque part une fonction `monMax` avec cette **signature** : type des paramètres et type du résultat



# Déclaration de fonctions

## Syntaxe

[compilation-separee/max.h](#)

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

## Sémantique

- ▶ Le programme **défini** quelque part une fonction `monMax` 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 monMax(int a, int b);
```

## Sémantique

- ▶ Le programme **définit** quelque part une fonction `monMax` 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 monMax(int a, int b);
```

## Sémantique

- ▶ Le programme **définit** quelque part une fonction `monMax` 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)

# Déclaration de fonctions

## Syntaxe

[compilation-separee/max.h](#)

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

## Sémantique

- ▶ Le programme **défini** quelque part une fonction `monMax` 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)

## ♣ application

Deux fonctions qui s'appellent réciproquement

## Compilation séparée

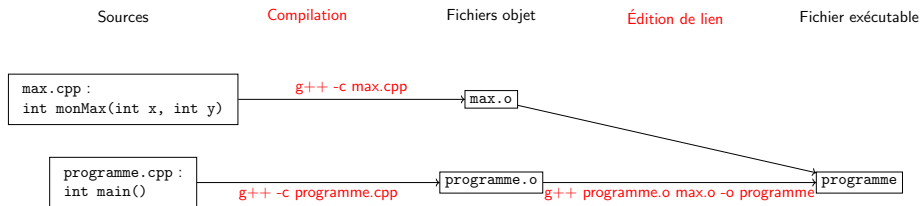
- ▶ Un programme peut être composé de plusieurs **fichiers source**  
Contenu :
  - ▶ Des **définitions** de fonctions
  - ▶ Des variables globales, ...

## 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 monMax(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 monMax(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 monMax(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 monMax(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 : 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
```

## Compilation

```
g++ -c max.cpp  
g++ -c programme1.cpp  
g++ max.o programme1.o -o programme1
```

En une seule étape :

```
g++ max.cpp programme1.cpp -o programme1
```

## Digression : surcharge de fonctions ♣

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

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

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

`max-surcharge.cpp`

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

`max-surcharge.cpp`

- ▶ À deux ou trois arguments (et plus?)

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

`max-surcharge.cpp`

## Digression : surcharge de fonctions ♣

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

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

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

`max-surcharge.cpp`

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

`max-surcharge.cpp`

- ▶ À deux ou trois arguments (et plus?)

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

`max-surcharge.cpp`

## 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;
}
```

- ▶ `using namespace std ;` : raccourcis pour `std`
- ▶ Vous verrez plus tard comment créer vos propres espaces de noms

## Digression : débogage, prévention

### Développement incrémental

- ▶ Toujours être « proche de quelque chose qui marche »

## Digression : débogage, prévention

### Développement incrémental

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

# Digression : 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



# Digression : 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**

# Résumé de la séance

## Programmes et compilation

- ▶ Programme, sources, binaire/assembleur
- ▶ Interpréteur, Compilateur
- ▶ Cycle de vie d'un programme

## Compilation séparée pour la modularité

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

# Résumé de la séance

## Programmes et compilation

- ▶ Programme, sources, binaire/assembleur
- ▶ Interpréteur, Compilateur
- ▶ Cycle de vie d'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

## Programmes et compilation

- ▶ Programme, sources, binaire/assembleur
- ▶ Interpréteur, Compilateur
- ▶ Cycle de vie d'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

## Programmes et compilation

- ▶ Programme, sources, binaire/assembleur
- ▶ Interpréteur, Compilateur
- ▶ Cycle de vie d'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