

# A <Basic> C++ Course

## 1 - Introduction

*Julien Deantoni*

# Contenue de l'UE (non contractuel)

- 1. Introduction, chaine de compilation, compilation séparée, exemple simple
- 2. Programmation procédurale, structuration de code et passage de paramètres
- 3. Allocation Dynamique de mémoire et constructeurs
- 4. Constructeurs, suite, destructeurs et surcharge d'opérateurs
- 5. Divers et Rappels
- 6. Templates en C++
- 7. Exemple complet de la classe document
- 8. POO part 1, explication au travers de la classe Document
- 9. POO part 2
- 10. POO part 3
- 11. Wrap-up
- 12. Lambda et programmation concurrente moderne

# Plan

- Du C++ ? pourquoi ?
- Principales différences avec java ?
  - le langage
  - La mise en œuvre (Interprété vs compilé vs mixte)
- From C to C++
- Programming and abstraction
  - Exemple simple en C et sa compilation
  - Exemple en C
  - Exemple simple en C++ et sa compilation
  - Exemple en C++

# Pourquoi C++ ?

- (encore) Très utilisé
  - <https://www.tiobe.com/tiobe-index/>

## TIOBE Index for September 2020



### September Headline: Programming Language C++ is doing very well

Back in 2003, the programming language C++ was a real winner. It peaked at 17.53% in August 2003, being close to the number #2 position and becoming winner of the programming language award of 2003. From then on C++ went downhill. After 2005 it didn't hit the 10% any more and in 2017 it scored an all time low of 4.55%. But if compared to last year, C++ is now the fastest growing language of the pack (+1.48%). I think that the new C++20 standard might be one of the main causes for this. Especially because of the new modules feature that is going to replace the dreadful include mechanism. C++ beats other languages with a positive trend such as R (+1.33%) and C# (+1.18%). On the other hand, Java is in real trouble with a loss of -3.18% in comparison to last year. - *Paul Jansen, CEO TIOBE Software*

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	15.95%	+0.74%
2	1	▼	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%

- Libre
- Performant

➔ Pourtant il souffre d'une mauvaise réputation comparé à JAVA

# Pourquoi C++ ?

- (encore) Très utilisé
  - <https://www.tiobe.com/tiobe-index/>
- Libre
- Performant
- ➔ Pourtant il souffre d'une mauvaise réputation comparé à JAVA

**Philosophie différente / complémentaire  
à Java : à connaître**

# Différences avec JAVA

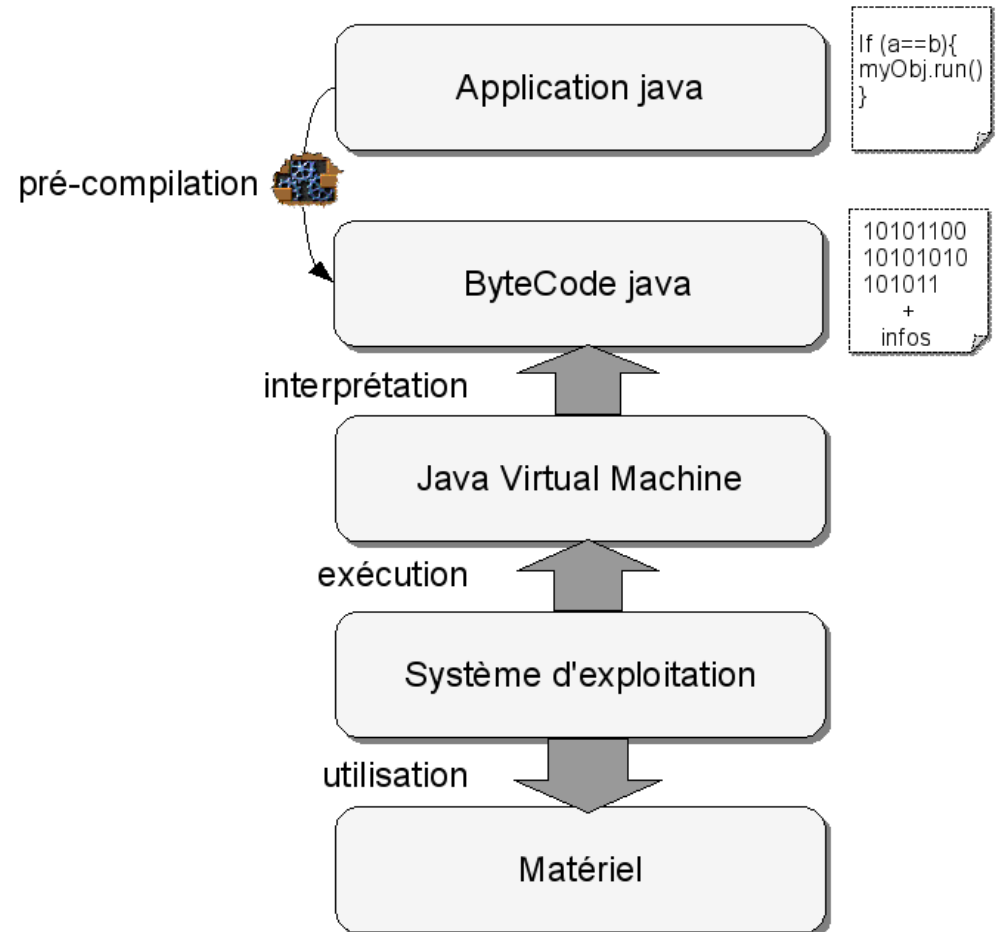
## *le langage*

- JAVA est plus simple
  - gestion automatique de la mémoire
  - IDE mieux fait ?
- mais aussi plus limité que C++
  - Pas de gestion fine de la mémoire,
  - Pas de surcharge des opérateurs,
  - Pas d'héritage multiple
  - Pas d'héritage privé
  - Mécanismes orientés objets imposés
  - Performance des IO
  - Peu de gestion fine de la synchronisation des threads
  - ...

# Différences avec JAVA

## *la mise en oeuvre*

- JAVA est pré-compilé puis interprété
- C++ est compilé

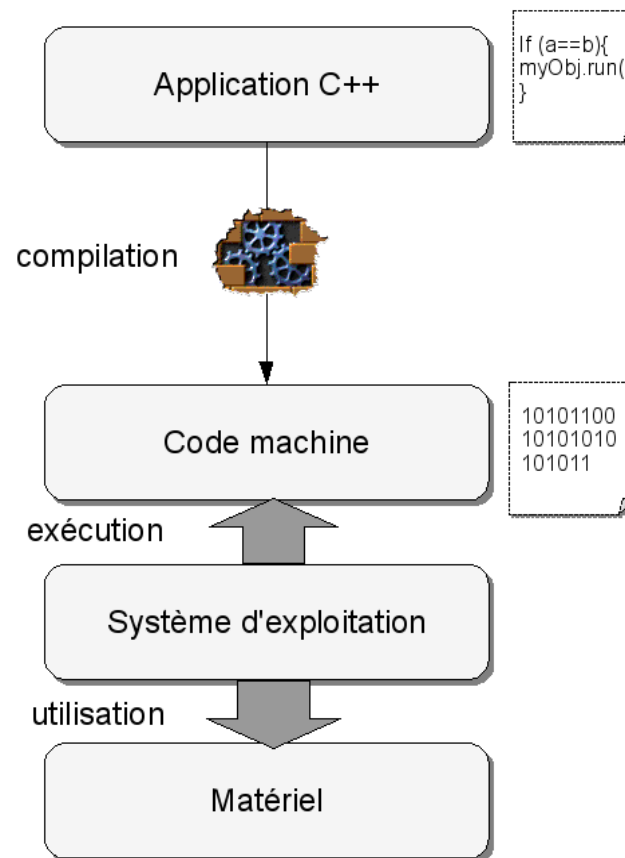


Noter que depuis quelques années, Java et son évolution sont dirigés par Oracle...

# Différences avec JAVA

## *la mise en oeuvre*

- JAVA est pré-compilé puis interprété
- C++ est compilé





# From C to C++

## Programming and abstraction

- ▶ **Procedures (and functions)**
  - ▶ Stress on structuring control flow
  - ▶ Less concern for structuring data

## *Le langage C*

### ► **Pros**

- High level language (Pascal style)
- **User definable data types**
- Well-suited to system programming and Unix
- **Support separate compiling**
- **Portable**
- **Efficient compilers**

## *Le langage C*

### ► Pros

- High level language (Pascal style)
- **User definable data types**
- Well-suited to system programming and Unix
- **Support separate compiling**
- **Portable**
- **Efficient compilers**

### ► Cons

- **Weak typing**
- Incomplete
  - no IO statements
  - no parallel statements
  - no string manipulation statements
  - no complex number, no global array manipulation...
- **Impossible to extend**
- Old fashioned approach to separate compiling and modularity

## Strong typing(1)

- ▶ ANSI C strong typing
  - ▶ Every object (constant, variable, function) has a type which must be known before the object is used
  - ▶ The compiler performs type checking (static analysis)
  - ▶ C/C++ makes it possible to mix data types in case of “natural” conversions (e.g. integer to real)
  - ▶ The type of a function (its signature) is given by a prototype specifying
    - ▶ the number of arguments and their types
    - ▶ the type of the return value

**double cos(double);**

# From C to C++

## Strong typing(1)

- ▶ ANSI C strong typing
  - ▶ Every object (constant, variable, function) has a type which must be known before the object is used
  - ▶ The compiler performs type checking (static analysis)
  - ▶ C/C++ makes it possible to mix data types in case of “natural” conversions (e.g. integer to real)
  - ▶ The type of a **function** (its **signature**) is given by a **prototype** specifying
    - ▶ the number of arguments and their types
    - ▶ the type of the return value

**double cos(double);**



This is the **declaration** of a function. (but here it is not **defined**, yet)

# From C to C++

## Strong typing(2)

- ▶ Strong typing makes it possible to overload operators and functions
  - ▶ the same name for several functions distinguished by their signature
  - ▶ overloading is specific to C++

**double cos(double);**  
**complex cos(complex);**

# From C to C++

## Programming and abstraction

- ▶ **Procedures** (and functions)
  - ▶ Stress on structuring control flow
  - ▶ Less concern for structuring data
  
- ▶ **Modules**
  - ▶ Group together data and procedures manipulating the data
  - ▶ Access control, information hiding, encapsulation
  - ▶ Separate the specification (interface) from the implementation (body)

# From C to C++

## Programming and abstraction

- ▶ **Procedures** (and functions)
  - ▶ Stress on structuring control flow
  - ▶ Less concern for structuring data
- ▶ **Modules**
  - ▶ Group together data and procedures manipulating the data
  - ▶ Access control, information hiding, encapsulation
  - ▶ Separate the specification (interface) from the implementation (body)
- ▶ **Abstract data types**
  - ▶ The module is turned into a type
  - ▶ One may declare instances (i.e. objects) of this type

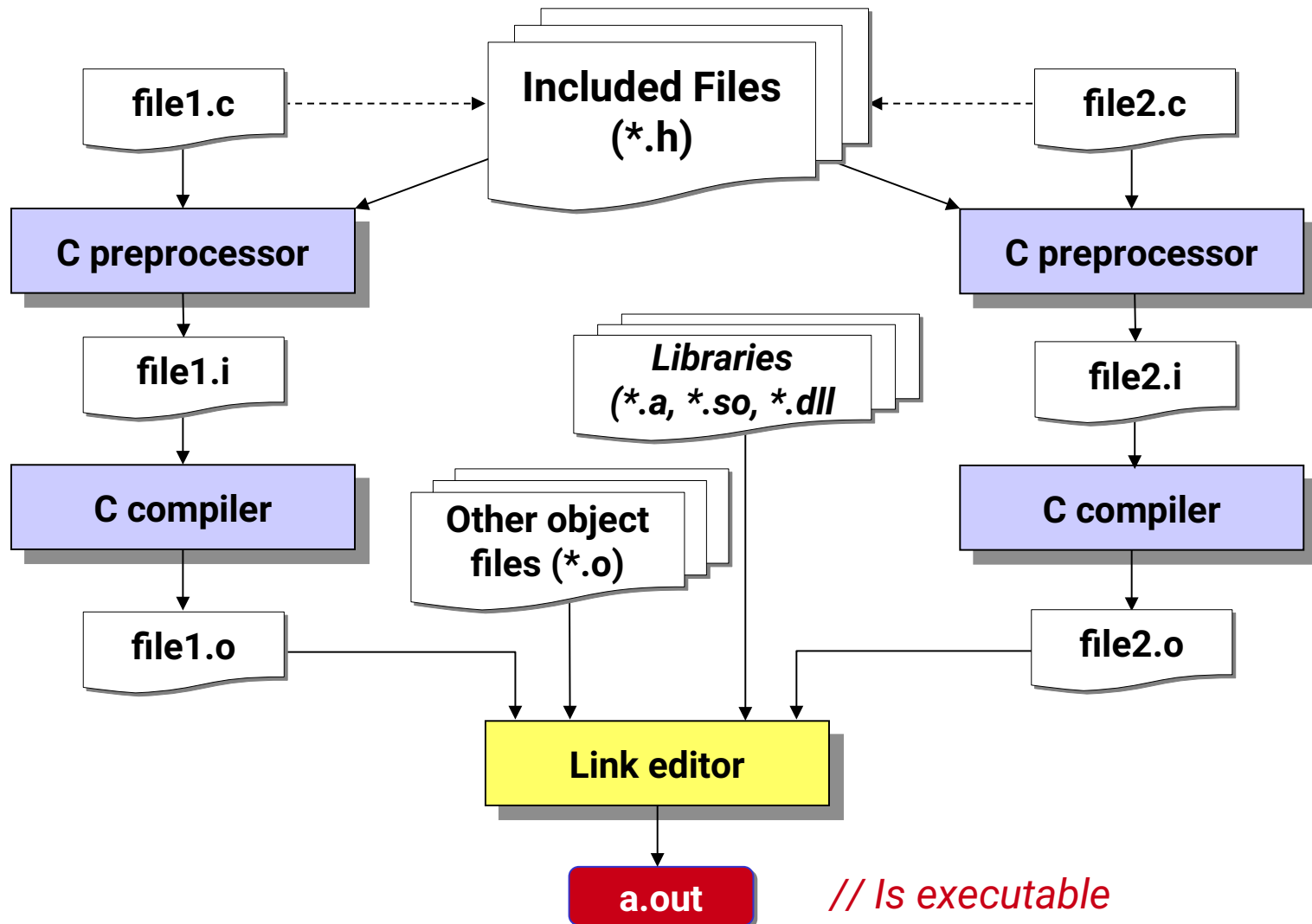


# From C to C++

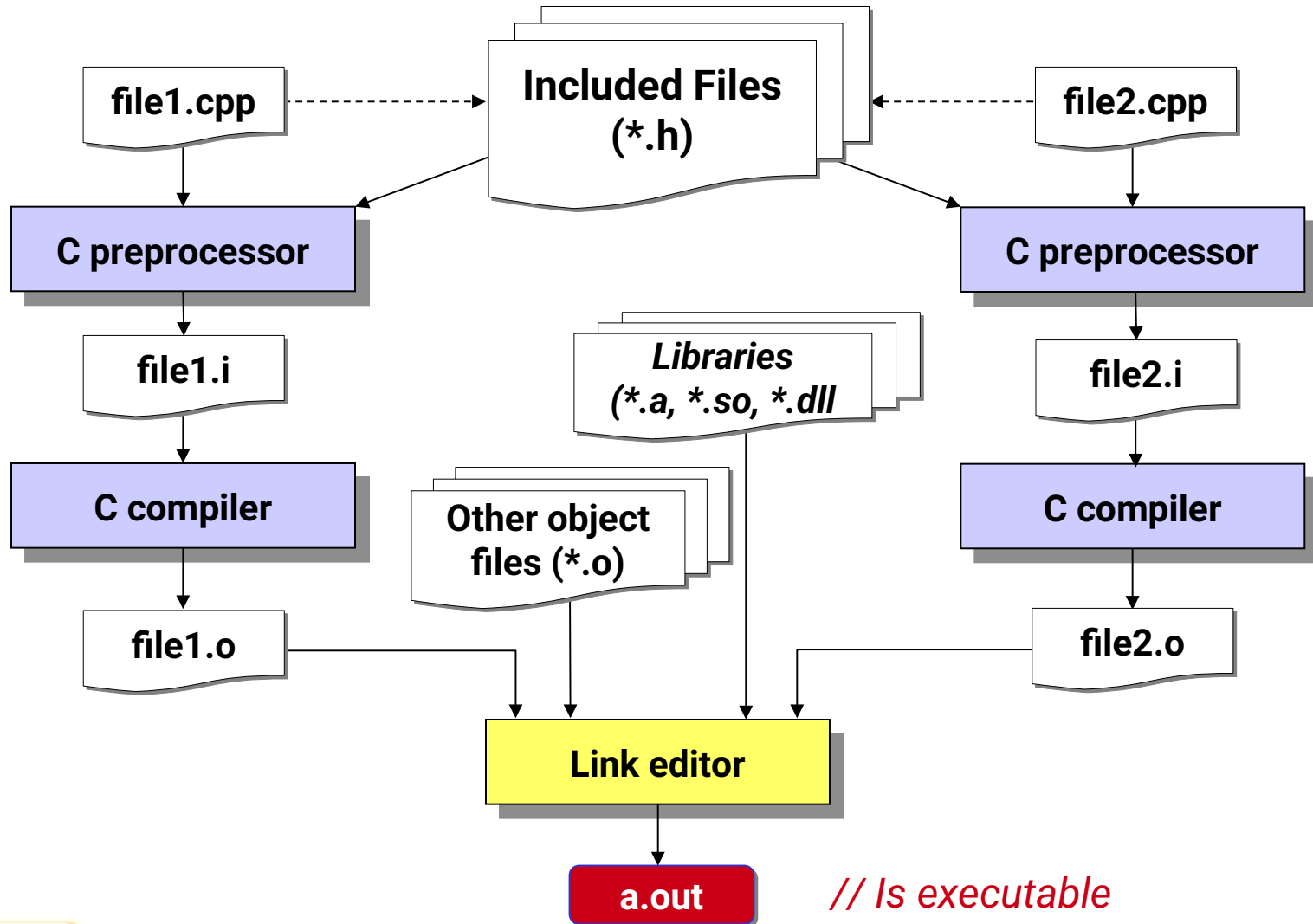
## Programming and abstraction

- ▶ **Procedures** (and functions)
  - ▶ Stress on structuring control flow
  - ▶ Less concern for structuring data
- ▶ **Modules**
  - ▶ Group together data and procedures manipulating the data
  - ▶ Access control, information hiding, encapsulation
  - ▶ Separate the specification (interface) from the implementation (body)
- ▶ **Abstract data types**
  - ▶ The module is turned into a type
  - ▶ One may declare instances (i.e. objects) of this type
- ▶ **Object-orientation**
  - ▶ Hierarchy among abstract data types (inheritance)
  - ▶ Dynamic (run-time) resolution of the exact type of an object (dynamic typing, late binding, polymorphism)

## compiler le C et le C++



## compiler le C et **le C++**



# Programming and abstraction

## *Le langage C* *Procédures et fonctions*

# Programming and abstraction

## Exemple simple en *langage C*

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    printf("Begin of main function ");
```

```
    return 0;
```

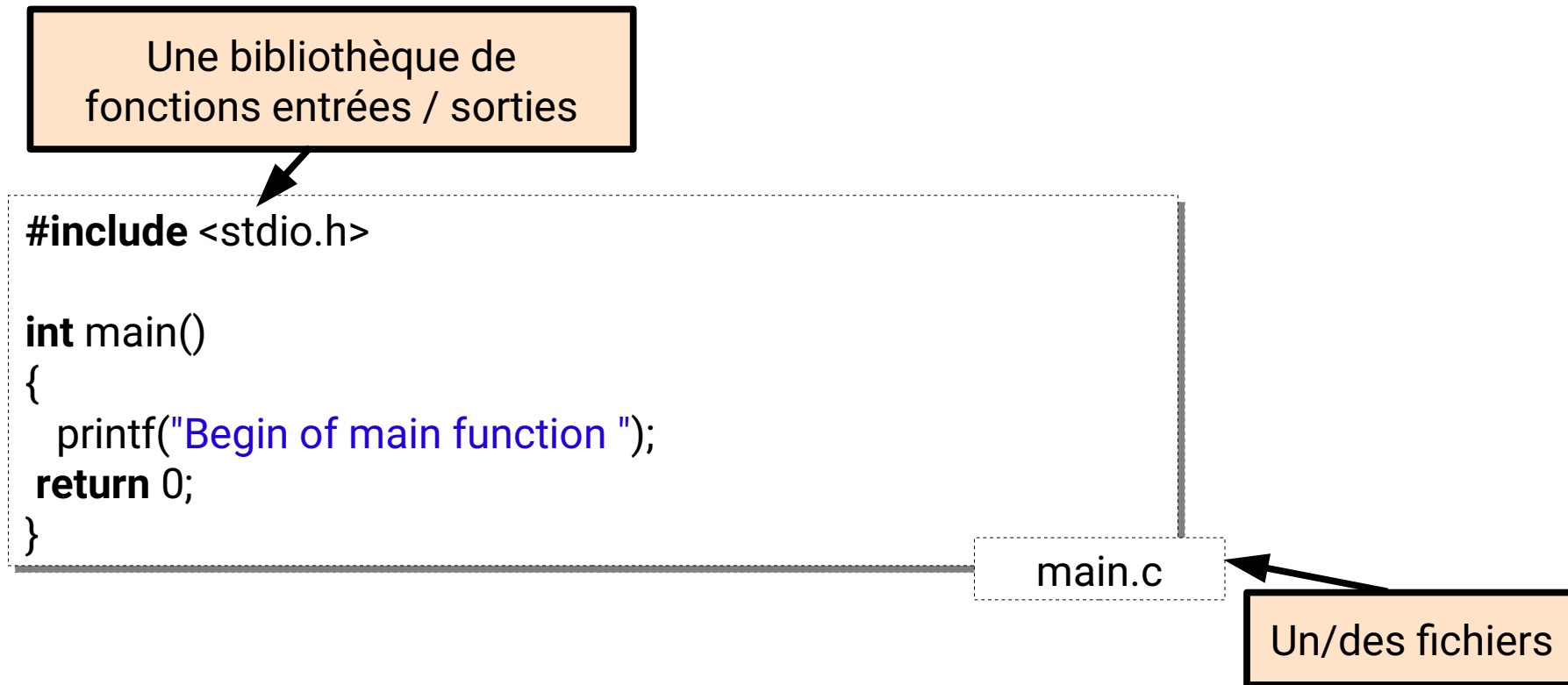
```
}
```

main.c

Un/des fichiers

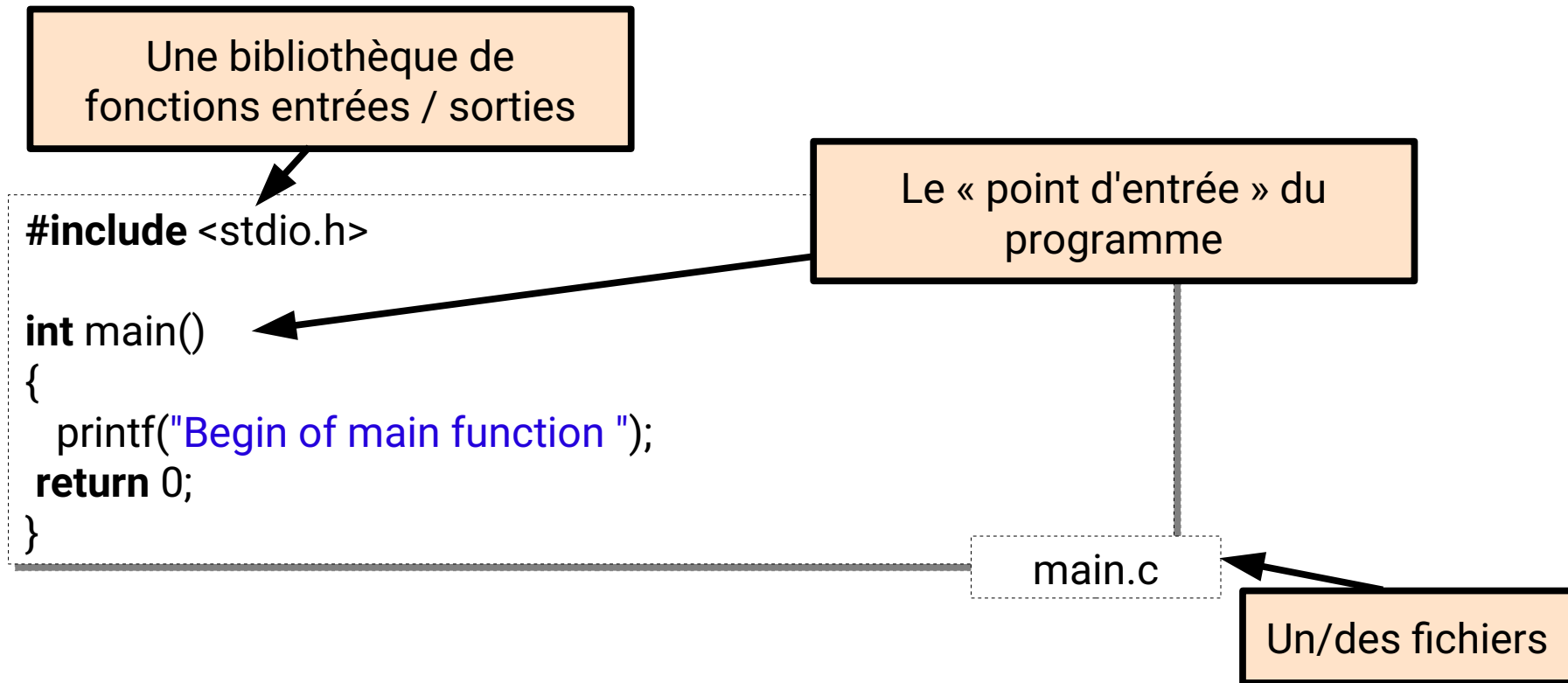
# Programming and abstraction

## Exemple simple en *langage C*



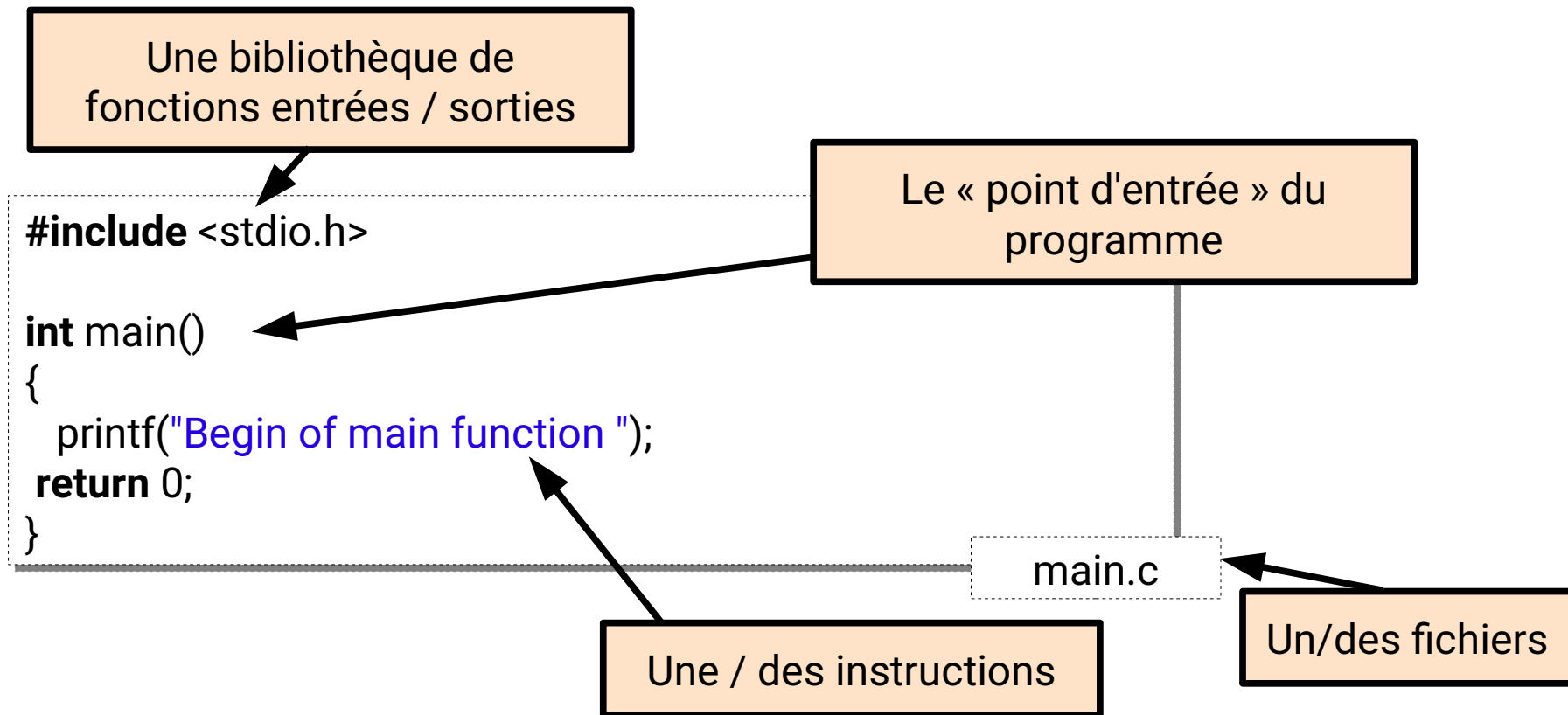
# Programming and abstraction

## Exemple simple en *langage C*



# Programming and abstraction

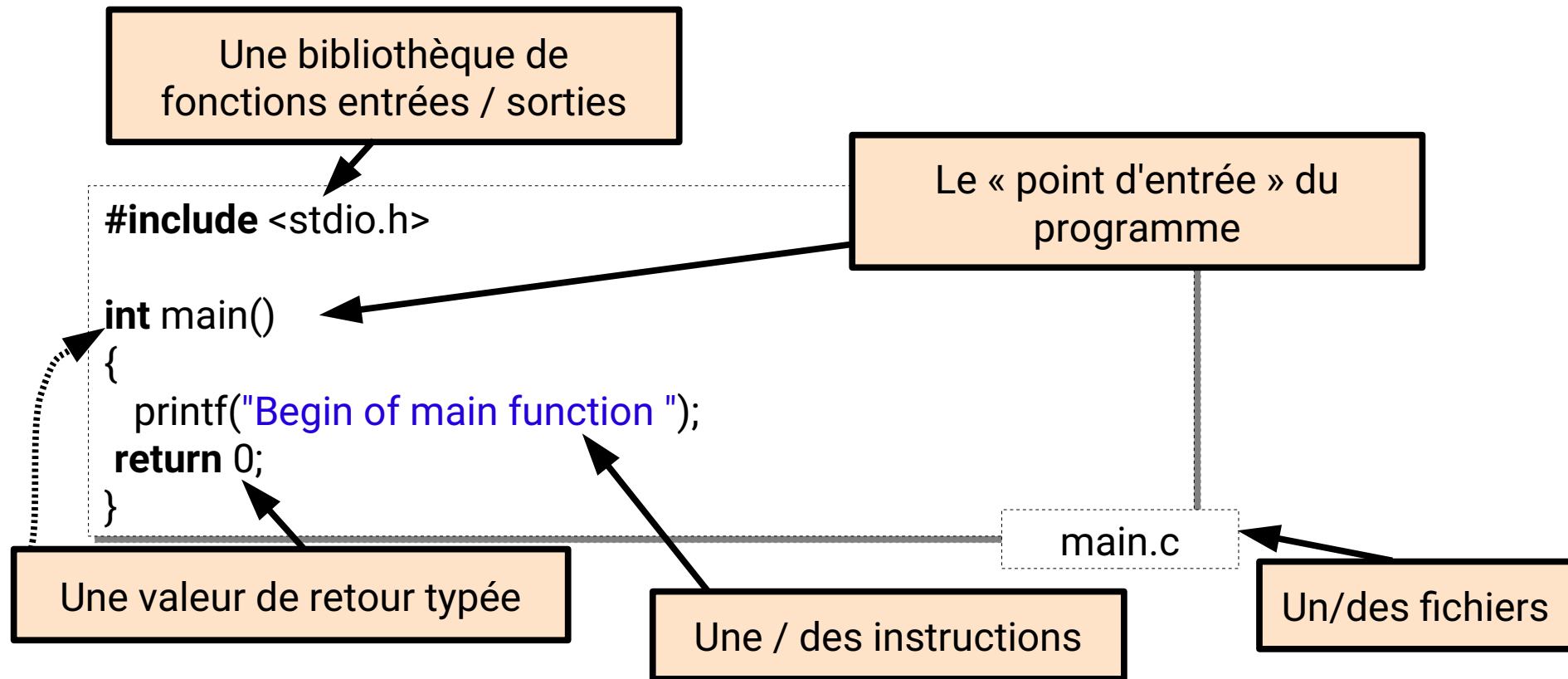
## Exemple simple en *langage C*



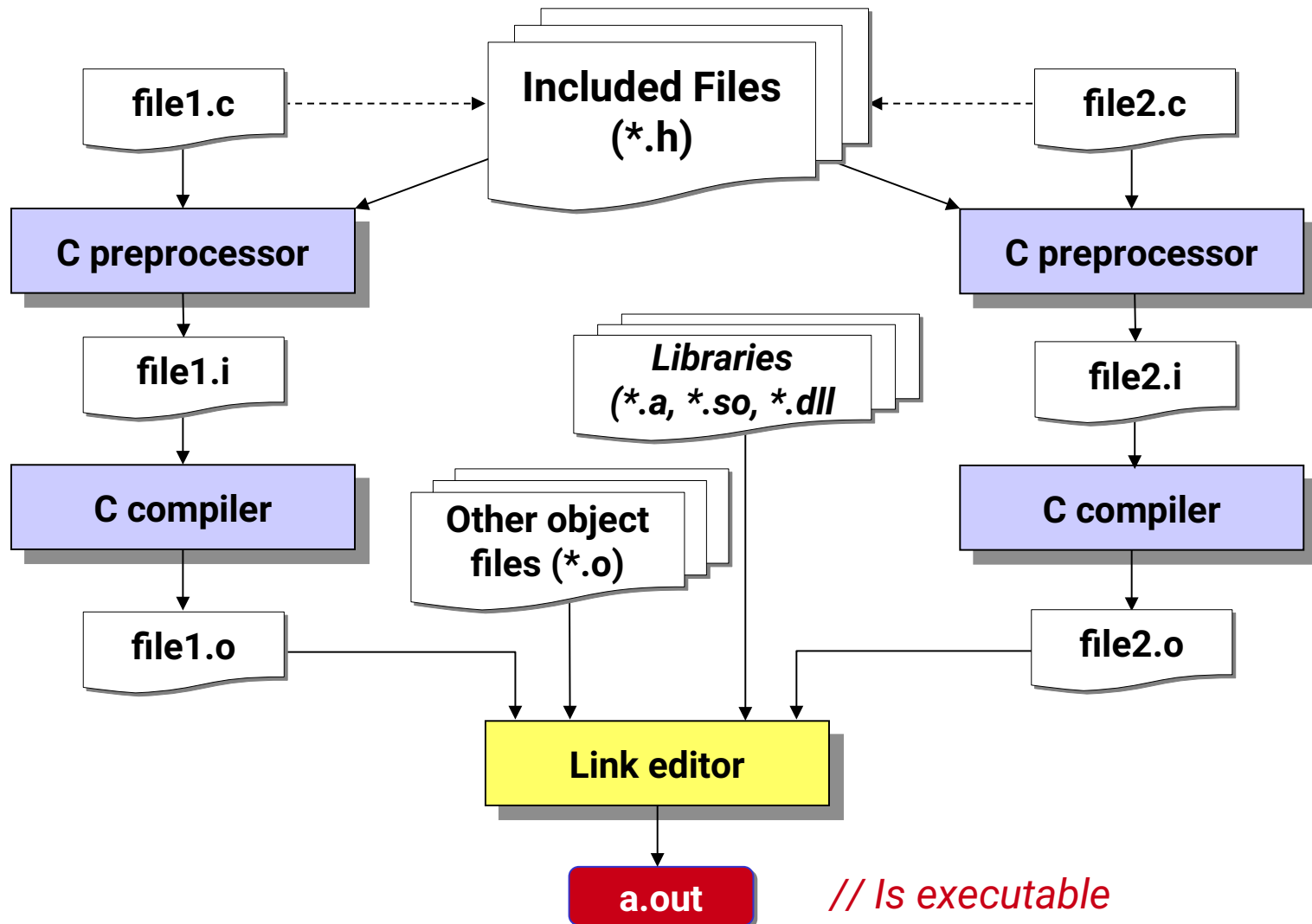


# Programming and abstraction

## Exemple simple en *langage C*



## *compiler le C*



# Programming and abstraction

## *compiler l'exemple simple en langage C*

```
#include <stdio.h>
```

```
int main()
```

```
{  
    printf("Begin of main function \n");  
    return 0;  
}
```

main.c

Compilation  
&  
création de main.o

Édition de lien  
&  
création de l'exécutable

Lancement de l'exécutable  
*i.e. lancement du main()*

```
jdeanton@FARCI:$ gcc -c main.c  
jdeanton@FARCI:$  
jdeanton@FARCI:$ gcc main.o -o executable  
jdeanton@FARCI:$  
jdeanton@FARCI:$ ./executable  
Begin of main function  
jdeanton@FARCI:$
```

# Programming and abstraction

## *compiler avec un « make » en langage C*

```
#include <stdio.h>
```

```
int main()  
{  
    printf("Begin of main function \n");  
    return 0;  
}
```

main.c

```
# sketchy makefile example
```

```
EXE_NAME=executable
```

```
LINK_C=gcc
```

```
COMPIL_C=gcc -c
```

```
example: main.o
```

```
    $(LINK_C) main.o -o $(EXE_NAME)
```

```
main.o: main.c
```

```
    $(COMPIL_C) main.c
```

Makefile

# Programming and abstraction

## *compiler avec un « make » en langage C*

```
#include <stdio.h>
```

```
int main()  
{  
    printf("Begin of main function \n");  
    return 0;  
}
```

main.c

```
# sketchy makefile example
```

```
EXE_NAME=executable
```

```
LINK_C=gcc
```

```
COMPIL_C=gcc -c
```

*Liste de sensibilité*

```
example: main.o
```

```
« \t » $(LINK_C) main.o -o $(EXE_NAME)
```

```
main.o: main.c
```

```
$(COMPIL_C) main.c
```

Makefile

# Programming and abstraction

## *compiler avec un « make » en langage C*

```
#include <stdio.h>
```

```
int main()  
{  
    printf("Begin of main function \n");  
    return 0;  
}
```

main.c

# sketchy makefile example

EXE\_NAME=executable

LINK\_C=gcc

COMPIL\_C=gcc -c

*cible*

*Liste de sensibilité*

example: main.o

« \t » \$(LINK\_C) main.o -o \$(EXE\_NAME)

main.o: main.c

\$(COMPIL\_C) main.c

Makefile

# Programming and abstraction

## *compiler avec un « make » en langage Cc*

```
# sketchy makefile example
```

```
EXE_NAME=executable
```

```
LINK_C=gcc
```

```
COMPIL_C=gcc -c
```

```
example: main.o
```

```
    $(LINK_C) main.o -o $(EXE_NAME)
```

```
main.o: main.c
```

```
    $(COMPIL_C) main.c
```

Makefile

```
jdeanton@FARCI:$ make example  
gcc -c main.c  
gcc main.o -o executable  
jdeanton@FARCI:$ ./executable  
Begin of main function  
jdeanton@FARCI:$
```

Lancement de la compilation  
de la cible « example »

# Programming and abstraction

## *compiler avec un « make » en langage Cc*

```
# sketchy makefile example
EXE_NAME=executable
LINK_C=gcc
COMPIL_C=gcc -c

example: main.o
    $(LINK_C) main.o -o $(EXE_NAME)

main.o: main.c
    $(COMPIL_C) main.c

clean:
    rm *.o $(EXE_NAME)
```

Makefile

Lancement de la compilation  
de la cible « clean »

```
jdeanton@FARCI:$ make clean
jdeanton@FARCI:$ ./executable
bash: ./executable: No such file or directory
jdeanton@FARCI:$
```

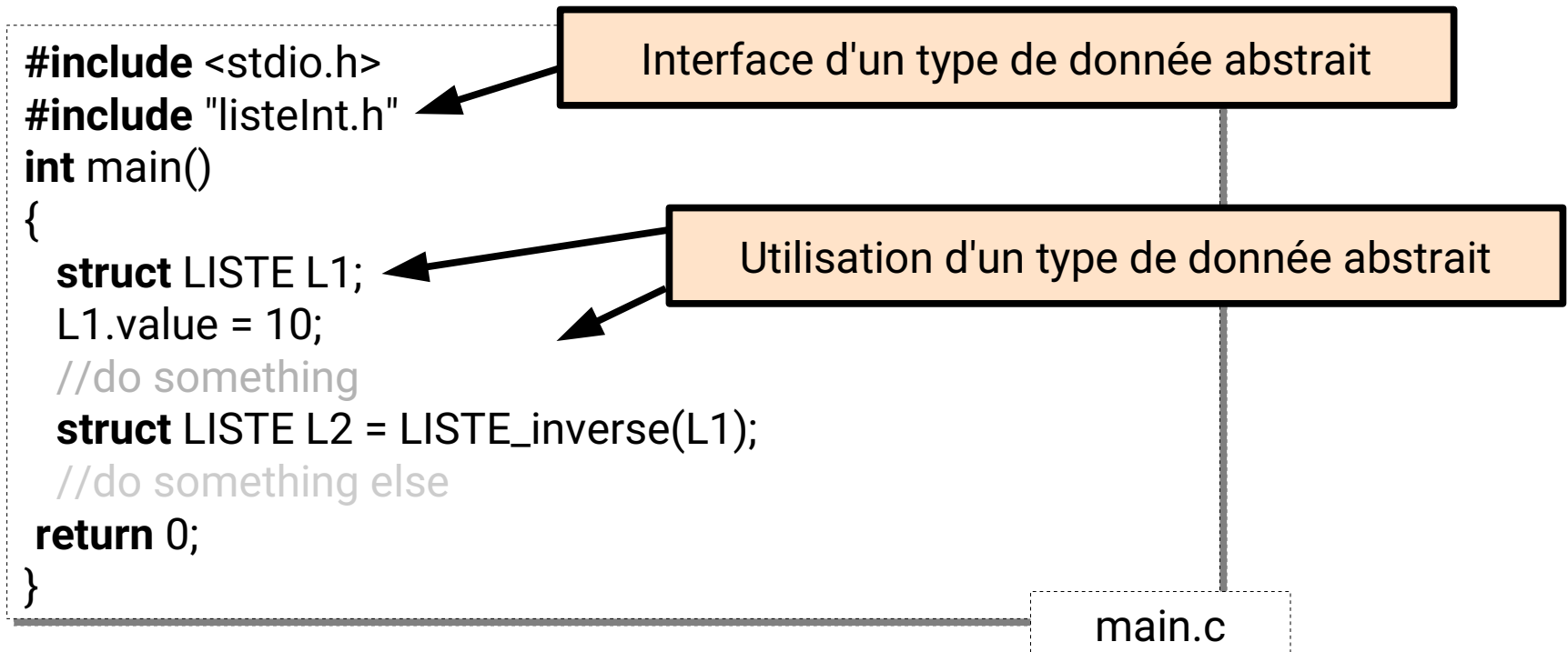


# Programming and abstraction

*Le langage C*  
*Les types de données abstraits*

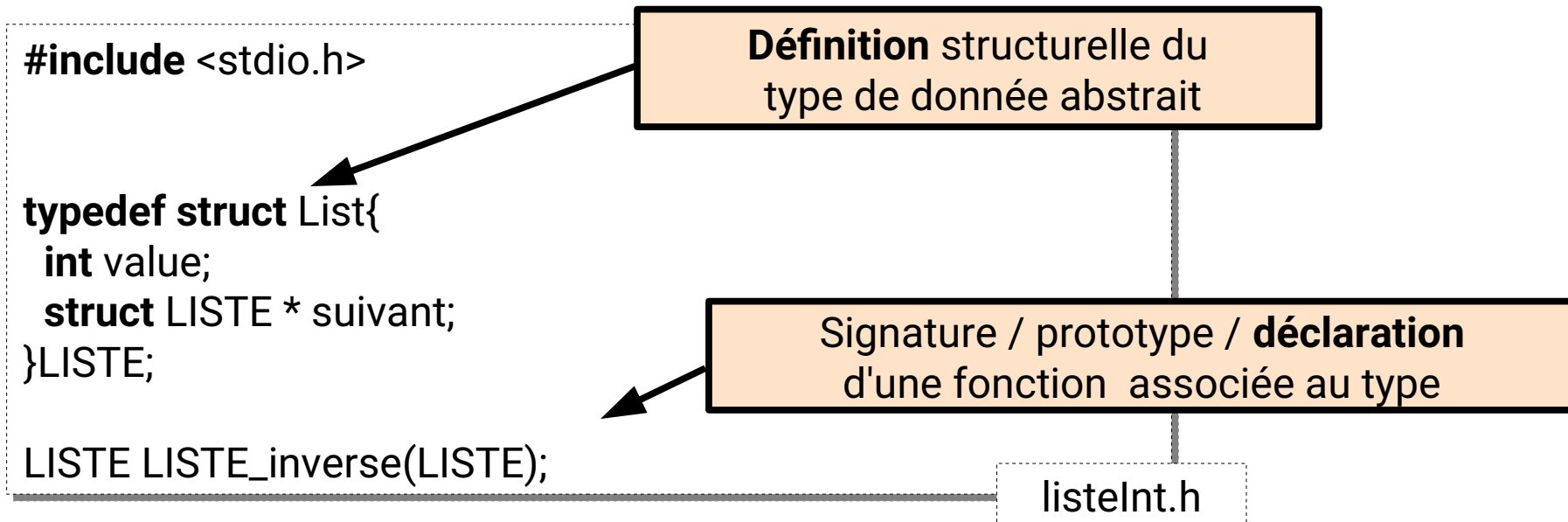
# Programming and abstraction

## Exemple en *langage C*



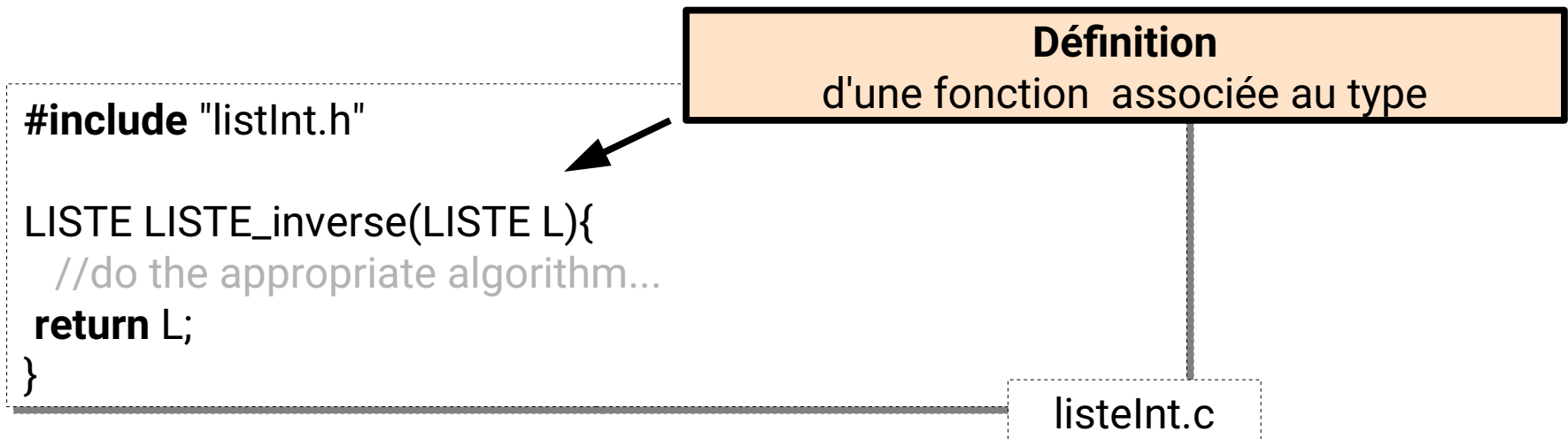
# Programming and abstraction

## Exemple en *langage C*



# Programming and abstraction

## Exemple en *langage C*



# Programming and abstraction

## *compiler avec un « make » en langage C*

# sketchy makefile example

EXE\_NAME=executable

LINK\_C=gcc

COMPIL\_C=gcc -c

example: main.o listInt.o

\$(LINK\_C) main.o listInt.o -o \$(EXE\_NAME)

main.o: main.c

\$(COMPIL\_C) main.c

listInt.o: listInt.c listInt.h

\$(COMPIL\_C) listInt.c

Makefile

# Programming and abstraction

## *un brin de méthodologie*

### ► Différents moments disjoints de la programmation :

#### ► La réalisation de type de données abstrait

- Déclarer une (des) structures(s) (+ les fonctions associées) → **.h**
- Implémenter la (les) fonctions associées(s) → **.c**

#### ► La réalisation du main

- Utiliser des types de données:
  - Qui sont prédéfinis (int, char, etc)
  - Que l'on a défini (les structures...)

# Programming and abstraction

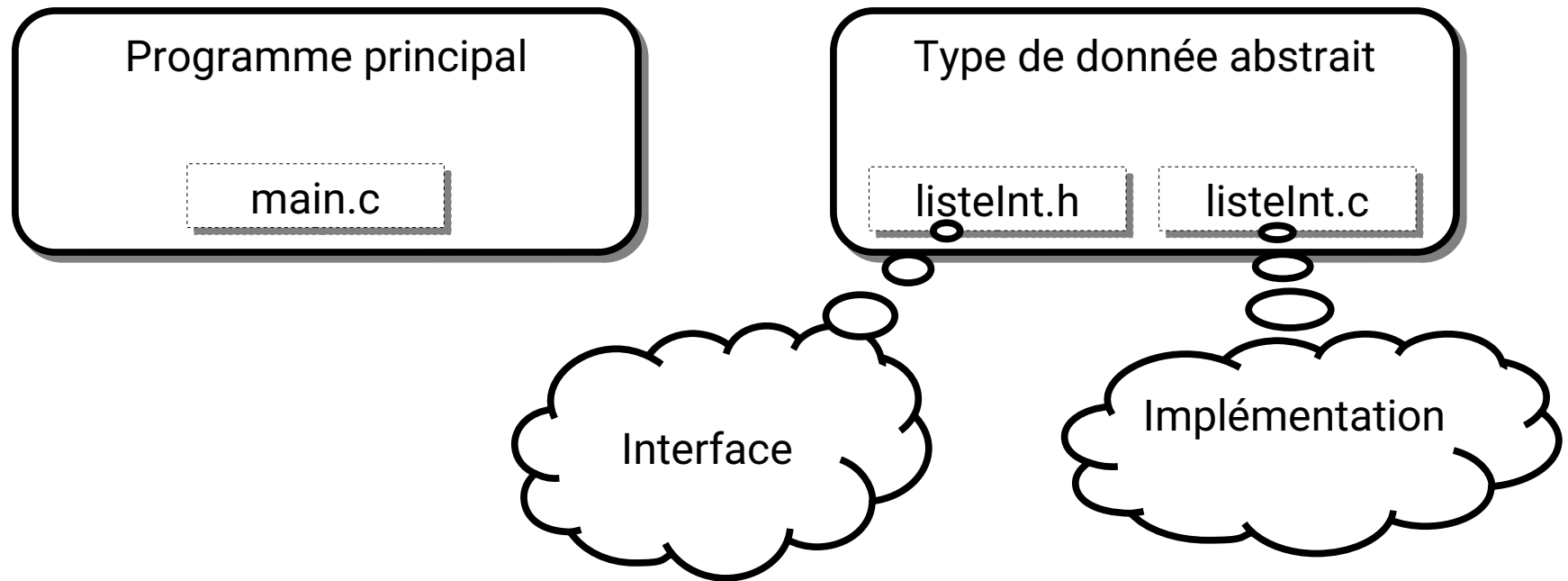
## *résumé de l'exemple en langage C*

Programme principal

main.c

# Programming and abstraction

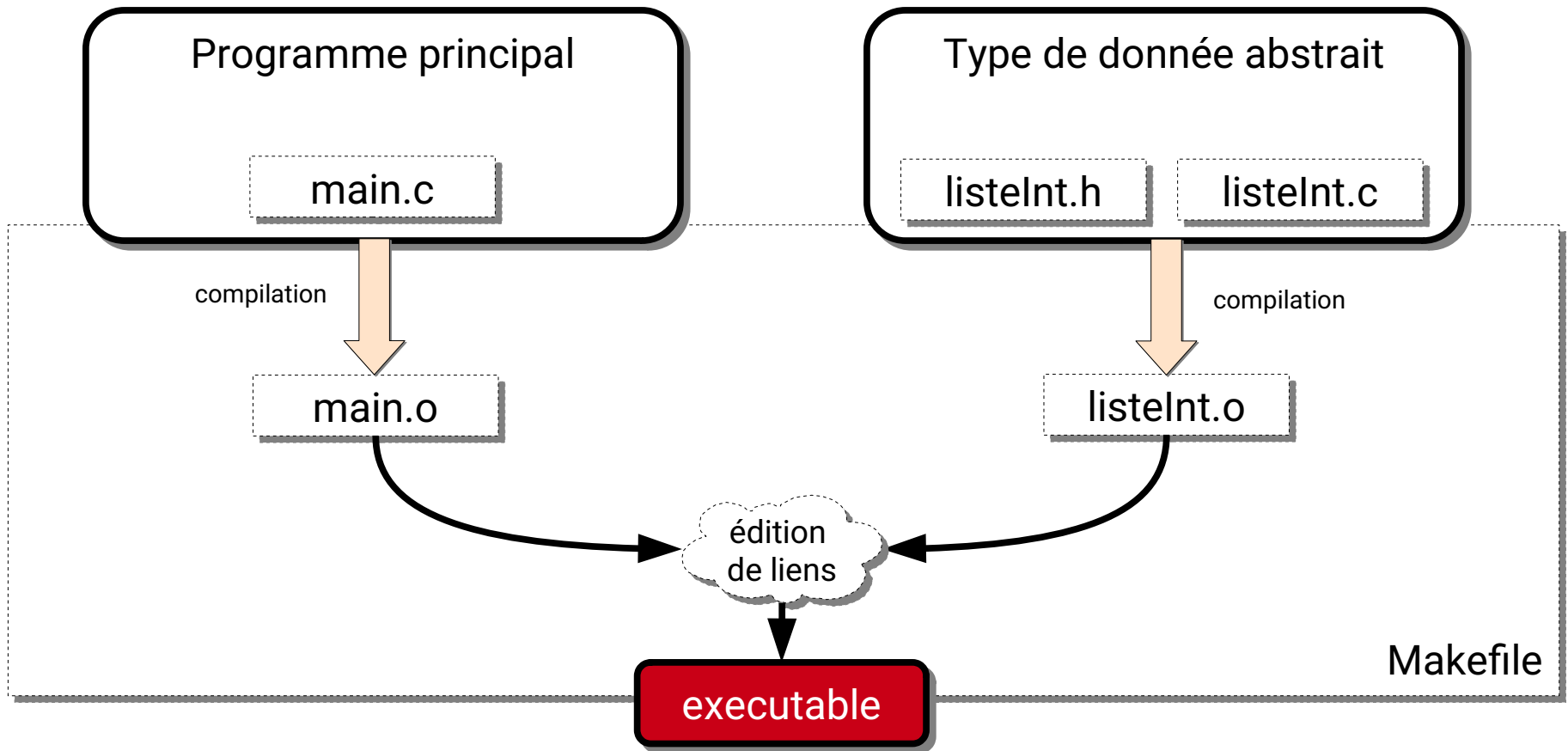
## *résumé de l'exemple en langage C*





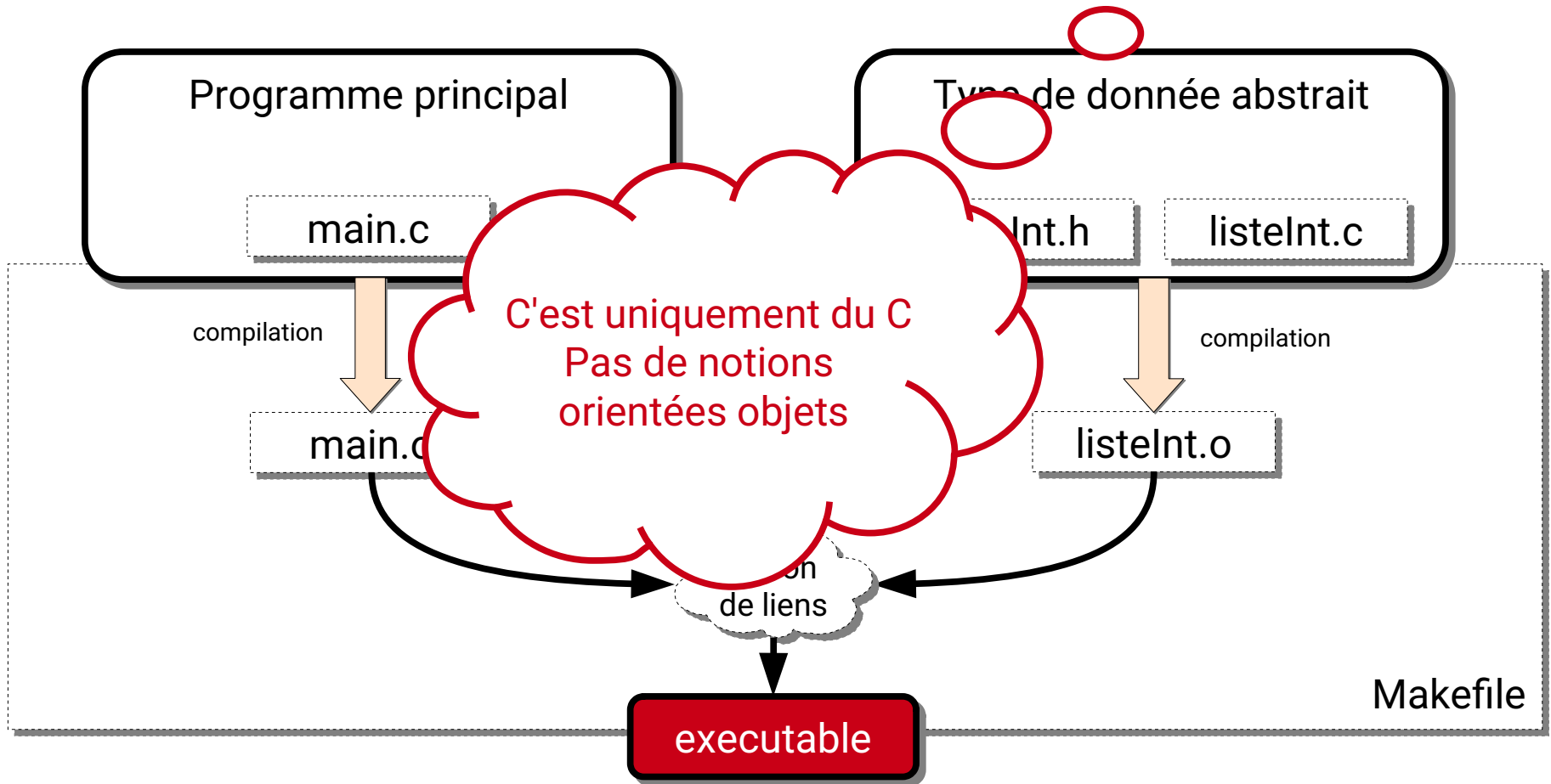
# Programming and abstraction

## résumé de l'exemple en langage C



# Programming and abstraction

## résumé de l'exemple en langage C



# Programming and abstraction

## *Le langage C++*

# Programming and abstraction

## *quoi de plus dans le C++ ?*

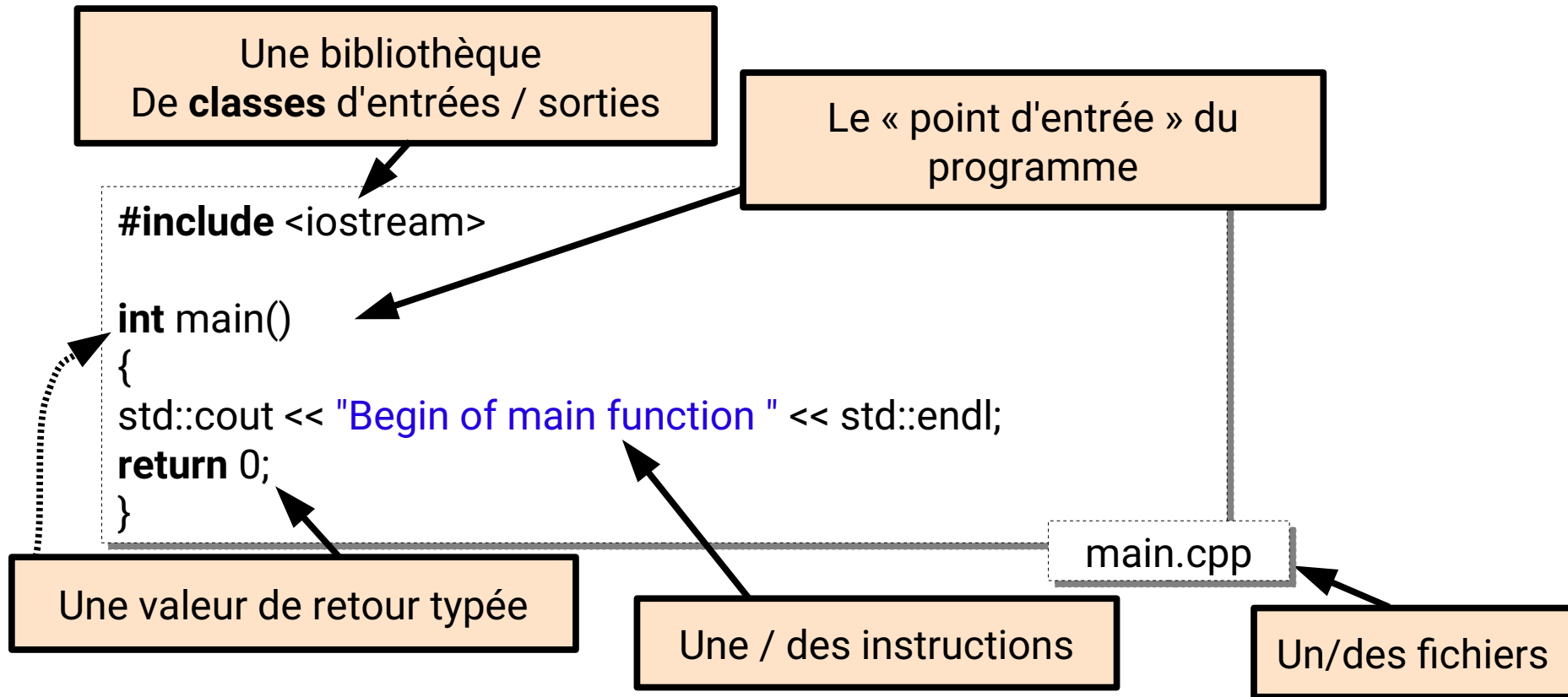
- ▶ Un typage fort
  - ▶ Surcharge de fonctions, d'opérateurs
- ▶ De la généricité : templates...
- ▶ La STL : Standard Template Library
  - ▶ Containers
  - ▶ Algorithms
  - ▶ ...
- ▶ Toutes les notions de ADT et orientées objets
  - ▶ Encapsulation / notion de visibilité
  - ▶ Hiérarchie entre les types de données (héritage)
  - ▶ Résolution dynamique (à l'exécution) du type effectif d'un objet (typage dynamique, polymorphisme)
- ▶ C++11
  - ▶ Lambda expressions, threads, inférence de type, ...

# Programming and abstraction

## *Le langage C++ Procédures et fonctions*

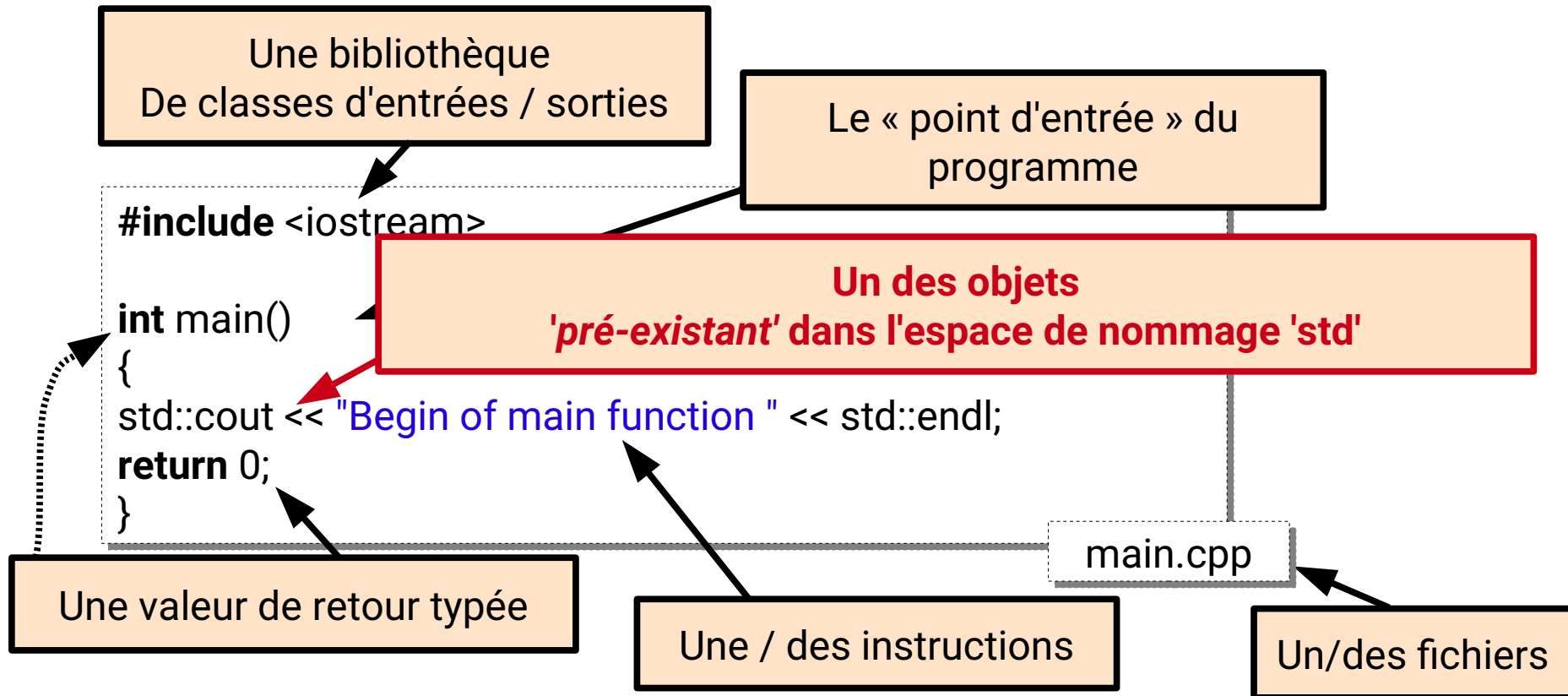
# Programming and abstraction

## Exemple simple en *langage C++*



# Programming and abstraction

## Exemple simple en *langage C++*



# Programming and abstraction

## *compiler avec un « make » en langage C*

```
#include <iostream>
```

```
int main()
{
    std::cout << "Begin of main function " << std::endl;
    return 0;
}
```

main.cpp

```
# sketchy makefile example
```

```
EXE_NAME=executable
```

```
LINK_CXX=g++
```

```
COMPIL_CXX=g++ -c
```

```
example: main.o
```

```
    $(LINK_CXX) main.o -o $(EXE_NAME)
```

```
main.o: main.cpp
```

```
    $(COMPIL_CXX) main.cpp
```

Makefile



# Programming and abstraction

## *compiler avec un « make » en langage C*

```
# sketchy makefile example
```

```
EXE_NAME=executable
```

```
LINK_CXX=g++
```

```
COMPIL_CXX=g++ -c
```

```
example: main.o
```

```
$(LINK_CXX) main.o -o $(EXE_NAME)
```

```
main.o: main.cpp
```

```
$(COMPIL_CXX) main.cpp
```

Makefile

```
jdeanton@FARCI:$ make example
```

```
g++ -c main.c
```

```
g++ main.o -o executable
```

```
jdeanton@FARCI:$ ./executable
```

```
Begin of main function
```

```
jdeanton@FARCI:$
```

Lancement de la compilation  
de la cible « example »

Lancement de l'exécutable  
*i.e. lancement du main()*

# Programming and abstraction

*Le langage C++  
Programmation orientée objet*

# Classe ou pas classe ?

► Différents moments disjoints de la programmation :

► La réalisation de classe

► La réalisation du main

## Classe ou pas classe ?

► Différents moments disjoints de la programmation :

► La réalisation de classe

► La réalisation du main

**Un code / programme ne contient pas forcément les deux !**

# Classe ou pas classe ?

## ► Différents moments disjoints de la programmation :

### ► La réalisation de classe

### ► La réalisation du main

#### ► Utiliser des classes :

- Qui sont Prédéfinies (par exemple celles de la STL)
- Que l'on a défini

# Classe ou pas classe ?

## ► Différents moments disjoints de la programmation :

### ► La réalisation de classe

- Déclarer une (des) classe(s) → **.h**
  - $\simeq$  faire le diagramme de classe
- Implémenter la (les) classe(s) → **.cpp**
  - $\simeq$  implémenter les opérations (diagrammes d'activités, stateCharts, diagrammes de séquence, etc)

### ► La réalisation du main

- Utiliser des classes :
  - Qui sont Prédéfinies (par exemple celles de la STL)
  - Que l'on a défini (voir point 1. )

# Utilisation de classes existantes

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

int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

main.cpp

# Utilisation de classes existantes

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

Inclusion de la bibliothèque  
associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

main.cpp



# Utilisation de classes existantes

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

Inclusion de la bibliothèque  
associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

Déclaration d'un  
Objet [+ initialisation]

main.cpp

# Utilisation de classes existantes

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

Inclusion de la bibliothèque  
associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;
}
```

Déclaration d'un  
Objet [+ initialisation]

## Pas de new en C++ !!!

(pour l'instant ;)

# Utilisation de classes existantes

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

Inclusion de la bibliothèque associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

Déclaration d'un  
Objet [+ initialisation]

Affichage de l'objet

main.cpp

# Utilisation de classes existantes

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

Inclusion de la bibliothèque associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

Déclaration d'un  
Objet [+ initialisation]

sérialisation de l'objet  
sur la sortie standard

main.cpp

# Utilisation de classes existantes

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

Inclusion de la bibliothèque associée à la classe

```
int main()
{
    std::cout << "Begin of main function " << std::endl;

    std::string aString="My first C++ string";
    std::cout << "\t"<< aString << std::endl;

    aString.append(" !!!!!!! ");
    std::cout << "\t"<< aString << std::endl;
    return 0;
}
```

Déclaration d'un  
Objet [+ initialisation]

sérialisation de l'objet  
sur la sortie standard

Appel d'une fonction membre  
de la classe *string*, sur l'objet

main.cpp

# To be continued