A <Basic> C++ Course

1 - Introduction

Julien Deantoni

Contenue de l'UE (non contractuel)

1. Introduction, chaine de compilation, compilation séparée, example 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. Example 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 August 2021

August Headline: Data Mining and AI languages are booming in the TIOBE index

Smart phones were the new hit many years ago. As a consequence, the programming languages that were used to write mobile applications became very popular as well. The best example of this is Objective-C, which peaked at position 3 in the TIOBE index, whereas it was only used to write apps for iPhones. Nowadays we have the same with data mining and AI. Programming languages in these fields are booming. The most striking example is Python that took over the second position from Java. Even old languages see a revival because of this, like the surge of Fortran. And, even more astonishing, we see Prolog re-entering the top 20 after 15 years... making an unexpected comeback. Prolog is used in IBM's Watson, one of the most well known AI engines. The only exception to all this is R, which is as opposed to the others, losing some positions. I guess Python is eating R's market share. Other interesting moves this month are: Rust from position #27 to #24 and Julia from position #35 to #26. Both Rust and Julia are strong candidates for a permanent top 20 position. -- *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.

Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	↑	Python	11.86%	+2.17%
3	2	↓	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5	5		C#	5.14%	+0.46%
6	6		Visual Basic	4.67%	+0.01%
7	7		JavaScript	2.95%	+0.07%

- 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

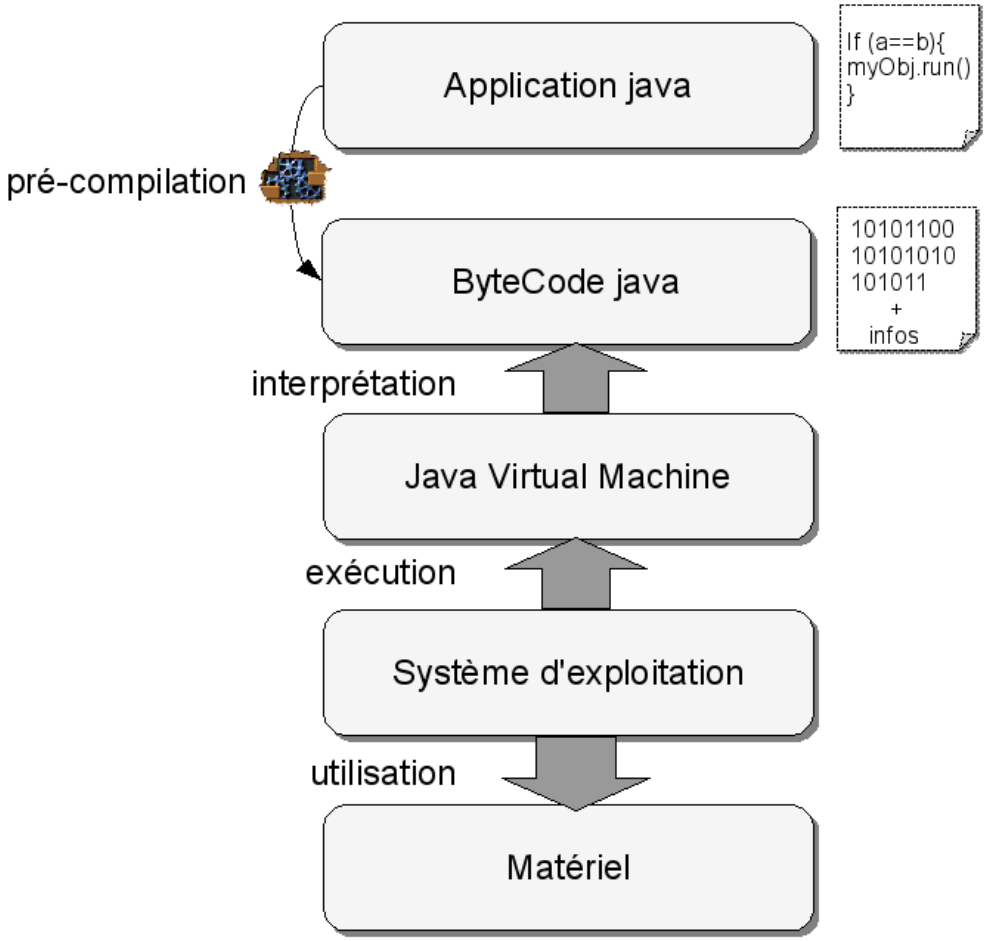
- JAVA est plus simple
  - gestion automatique de la mémoire (garbage collector)
  - IDE mieux fait ?
- mais aussi plus limité que C++
  - Pas de gestion fine de la mémoire,
  - Pas de surcharge des opérateurs,
  - Pas d'héritage multiple
  - Pas d'héritage privé
  - Mécanismes orientés objets imposés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

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

Différences avec JAVA

la mise en oeuvre

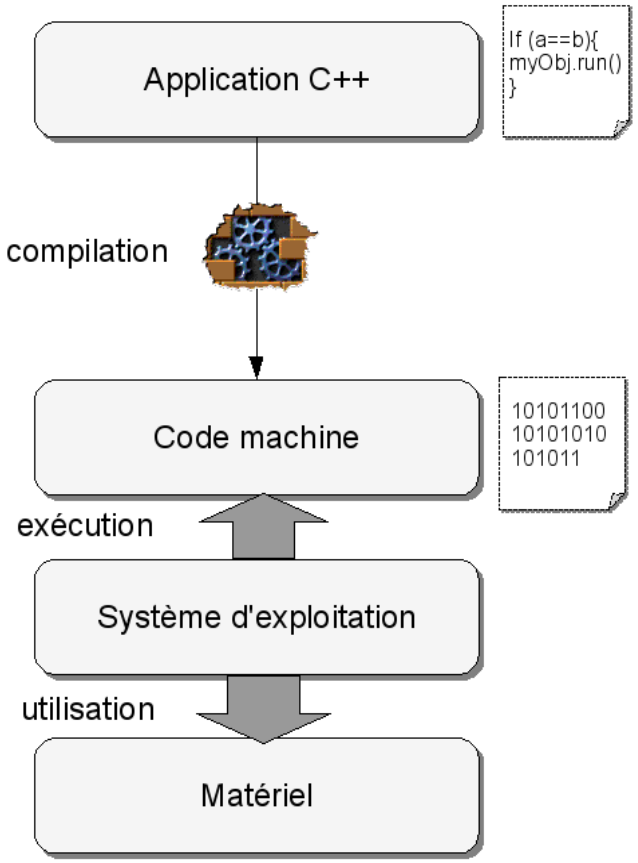
- JAVA est pré-compilé puis interprété
- C++ est compilé



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

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);

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)

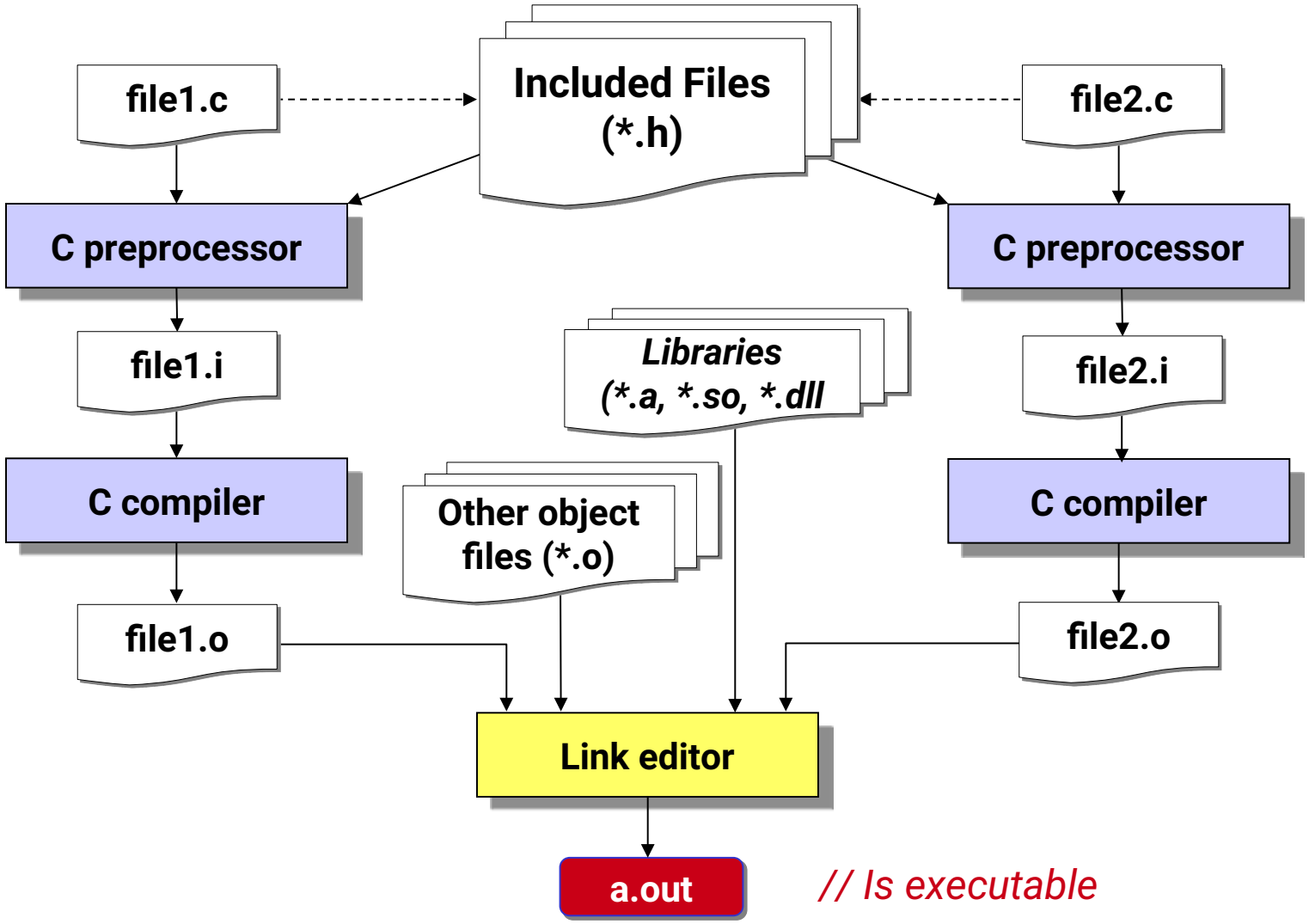
From C to C++
Programming and abstraction

- **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

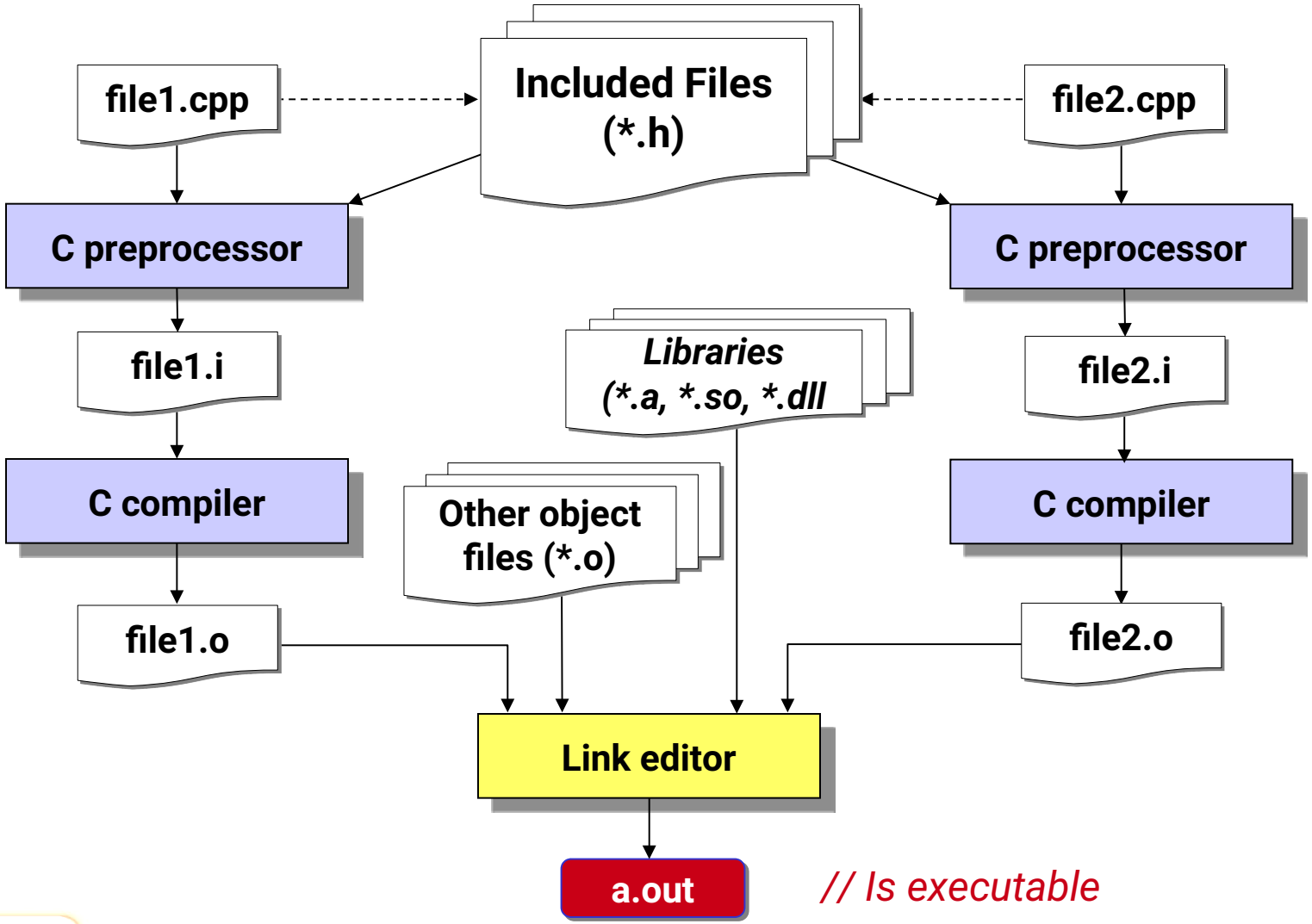
From C to C++
Programming and abstraction

- **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++



Le langage C
Procédures et fonctions

Exemple simple en *langage C*

```c
#include <stdio.h>

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

main.c

Un/des fichiers

Exemple simple en *langage C*

Une bibliothèque de fonctions entrées / sorties

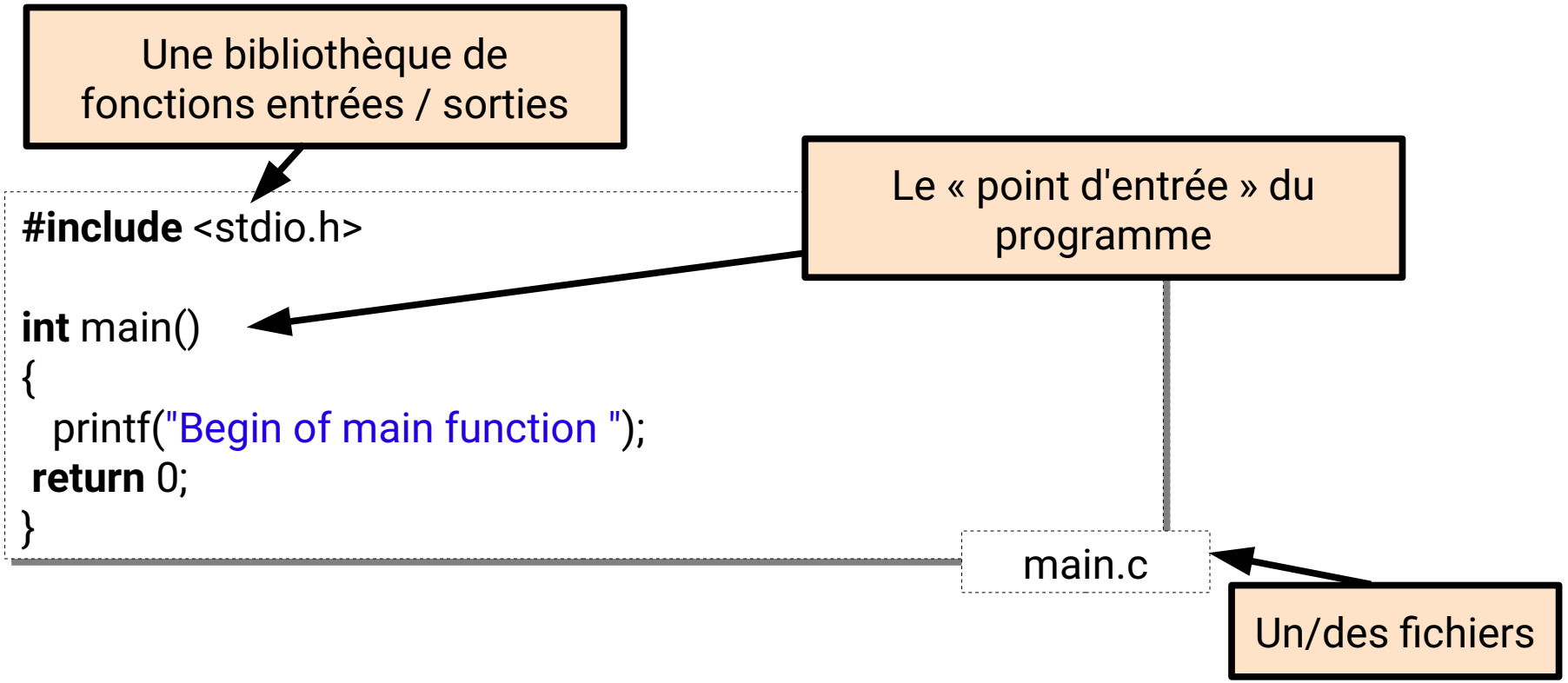
```c
#include <stdio.h>

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

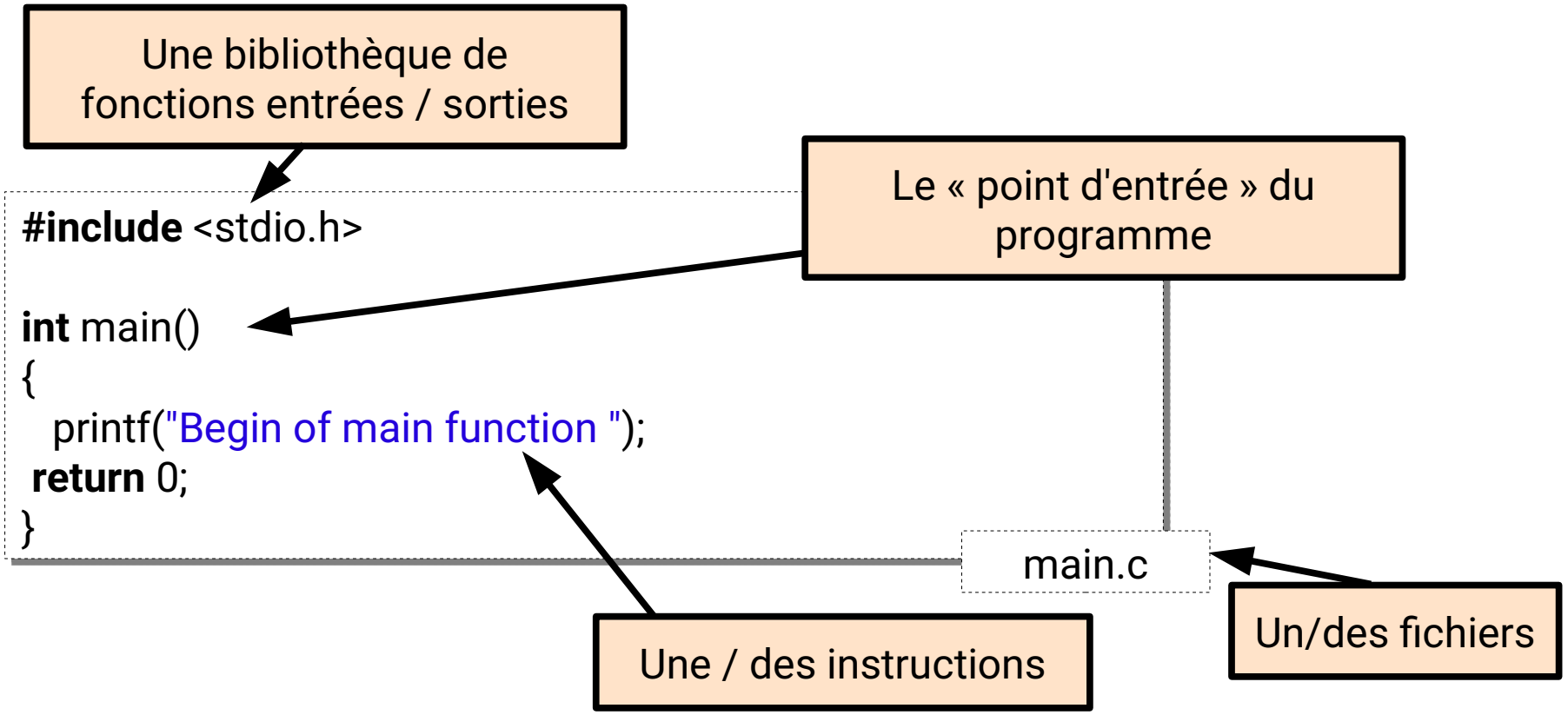
main.c

Un/des fichiers

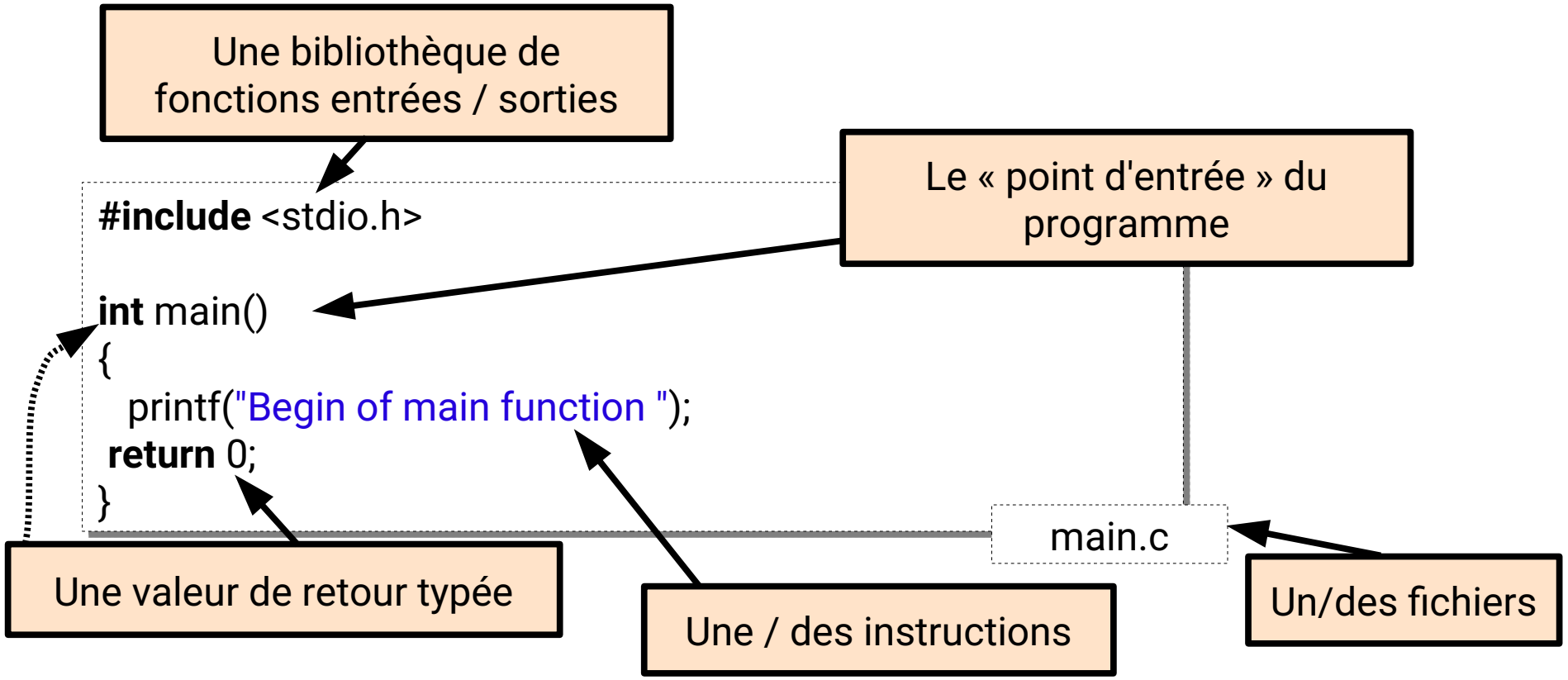
Exemple simple en *langage C*



Exemple simple en *langage C*

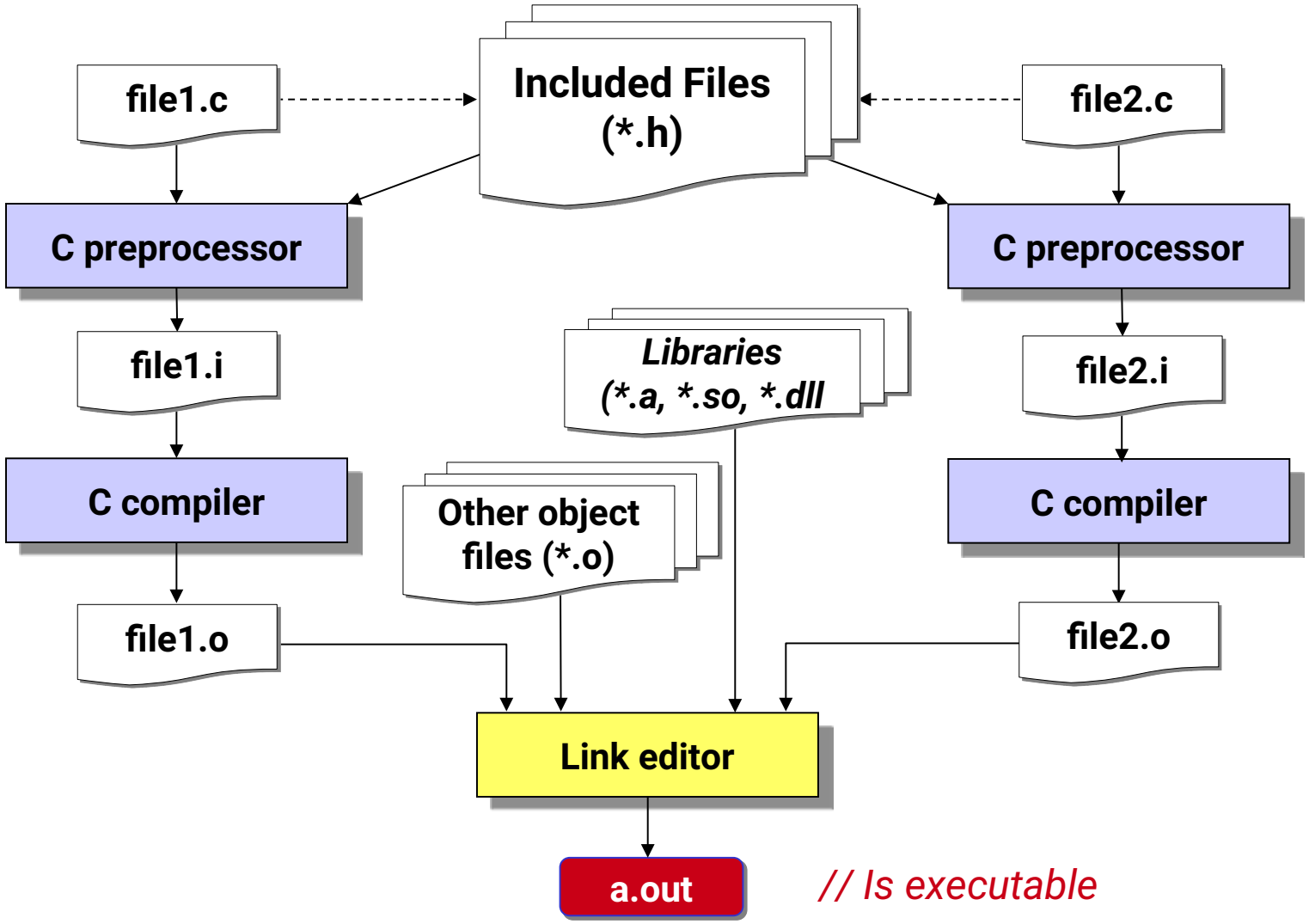


Exemple simple en *langage C*



Programming and abstraction

compiler le C



Programming and abstraction

compiler l'exemple simple en langage C

```c
#include <stdio.h>

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

main.c

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

Compilation
&
création de main.o

Édition de lien
&
création de l'exécutable

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

Programming and abstraction

compiler avec un « make » en langage C

```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

compiler avec un « make » en langage C

```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
```

Liste de sensibilité

« \t »

Makefile

Programming and abstraction

compiler avec un « make » en langage C

```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
```

cible

Liste de sensibilité

« \t »

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

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

Lancement de la compilation de la cible « clean »

Le langage C
Les types de données abstraits

Exemple en *langage C*

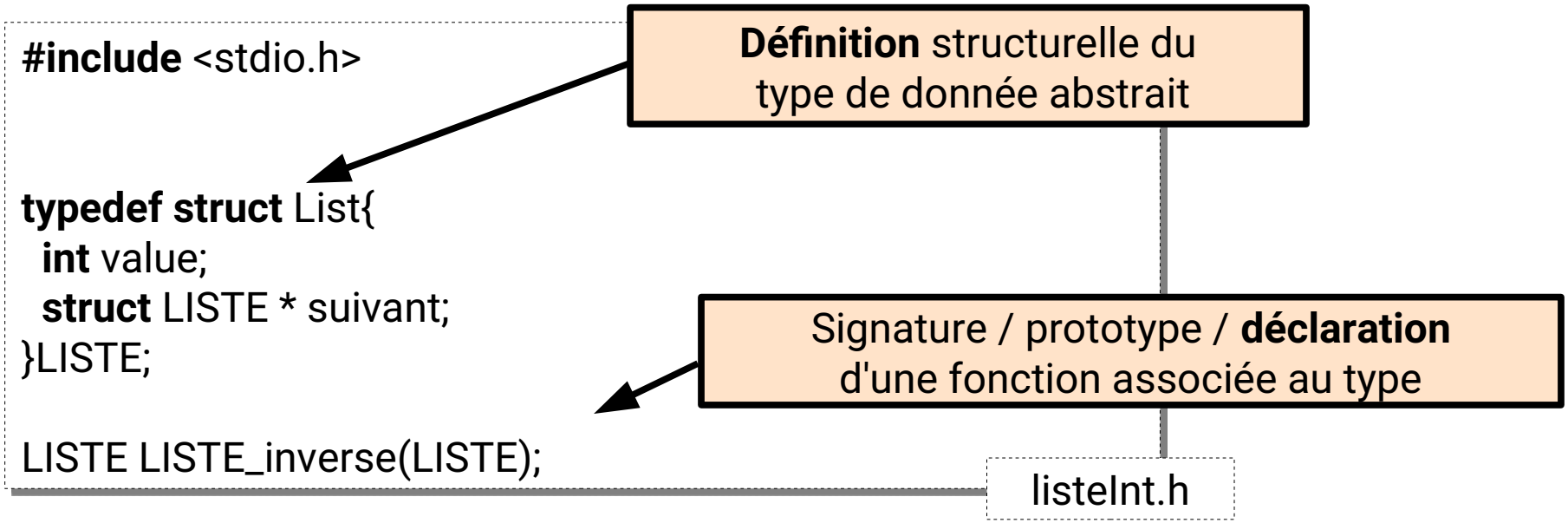
```c
#include <stdio.h>
#include "listeInt.h"
int main()
{
   struct LISTE L1;
   L1.value = 10;
   //do something
   struct LISTE L2 = LISTE_inverse(L1);
   //do something else
 return 0;
}
```

Interface d'un type de donnée abstrait

Utilisation d'un type de donnée abstrait

main.c

Exemple en *langage C*



Exemple en *langage C*

Définition
d'une fonction associée au type

```
#include "listInt.h"

LISTE LISTE_inverse(LISTE L){
  //do the appropriate algorithm...
 return L;
}
```

listeInt.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
```

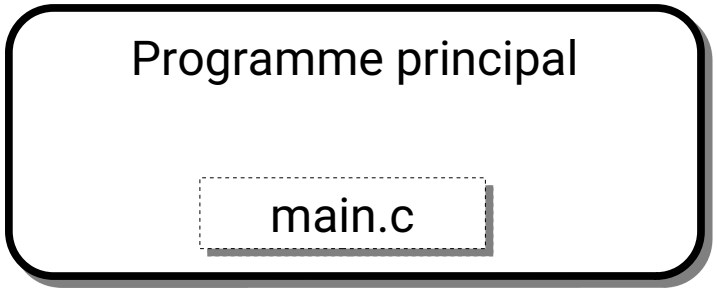
Makefile

Programming and abstraction

- 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...)

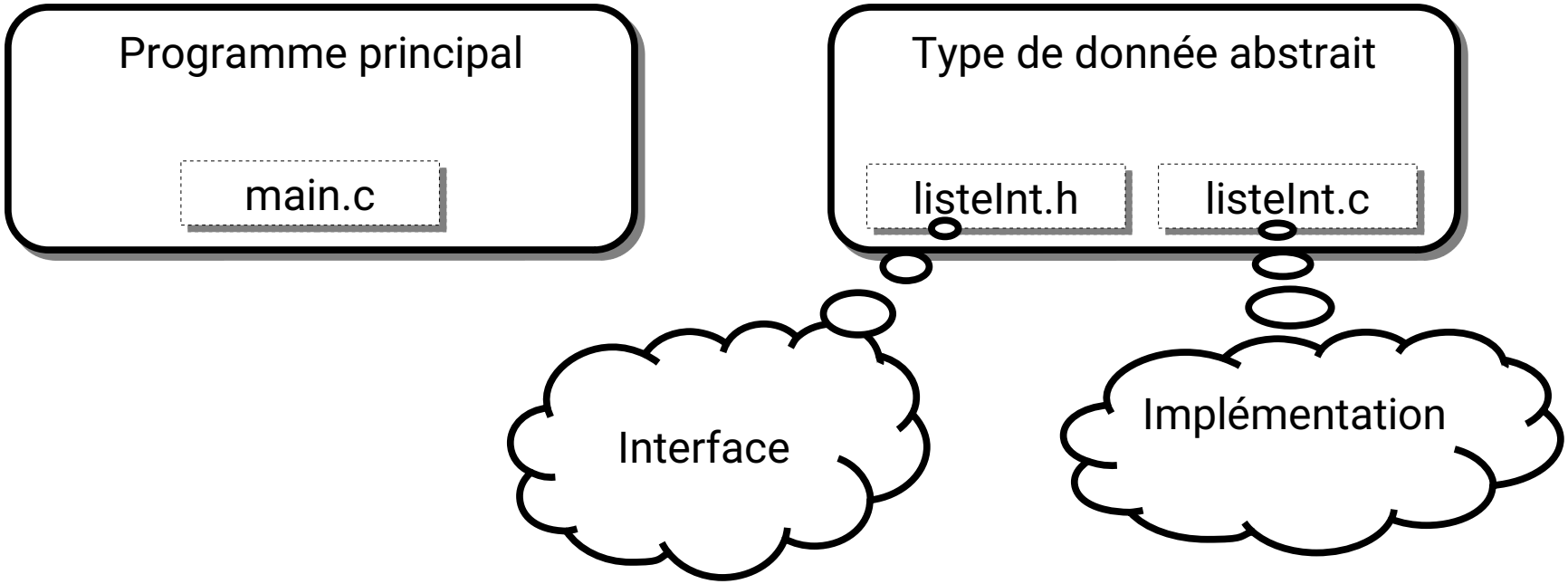
Programming and abstraction

resumé *de l'exemple en langage C*

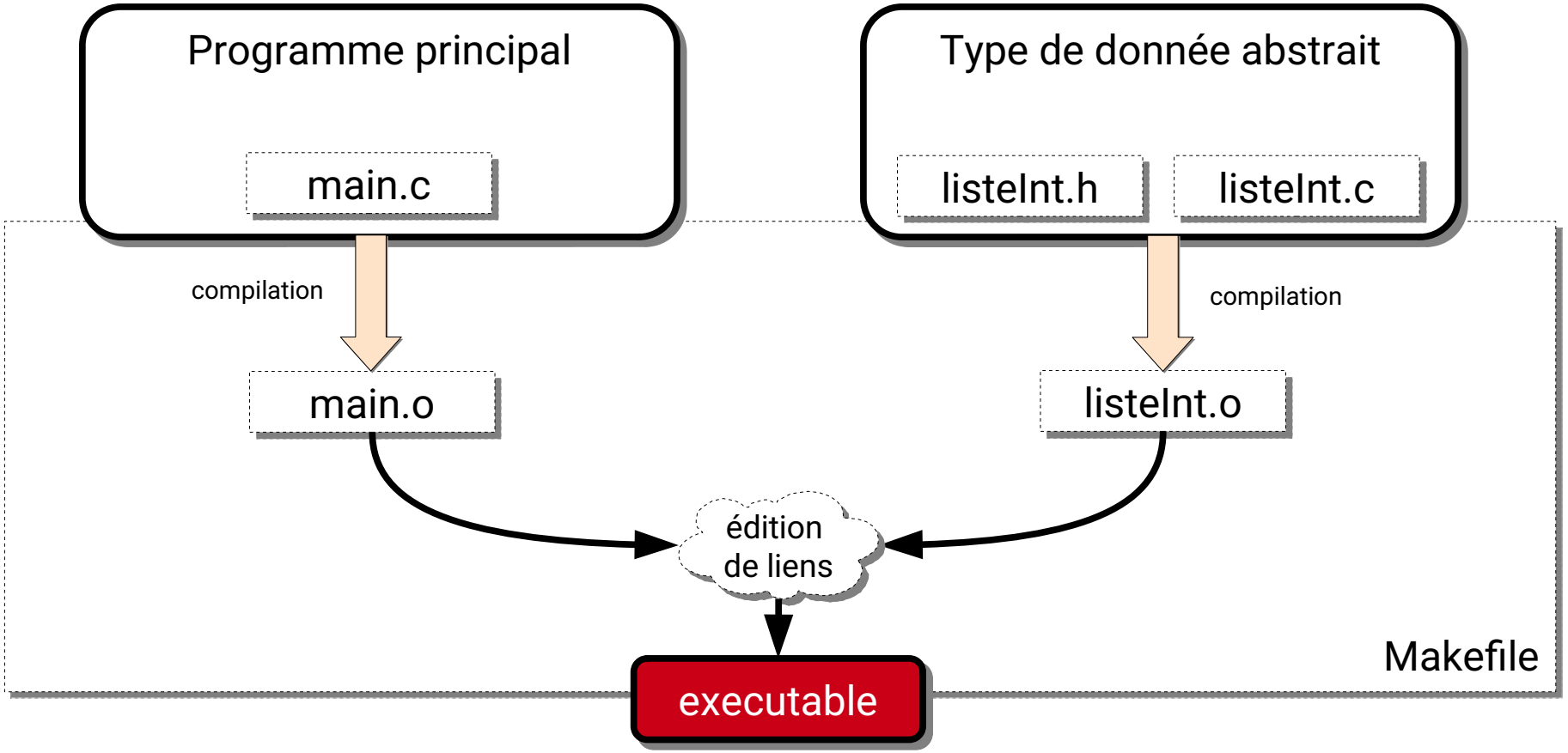


Programming and abstraction

resumé *de l'exemple en langage C*

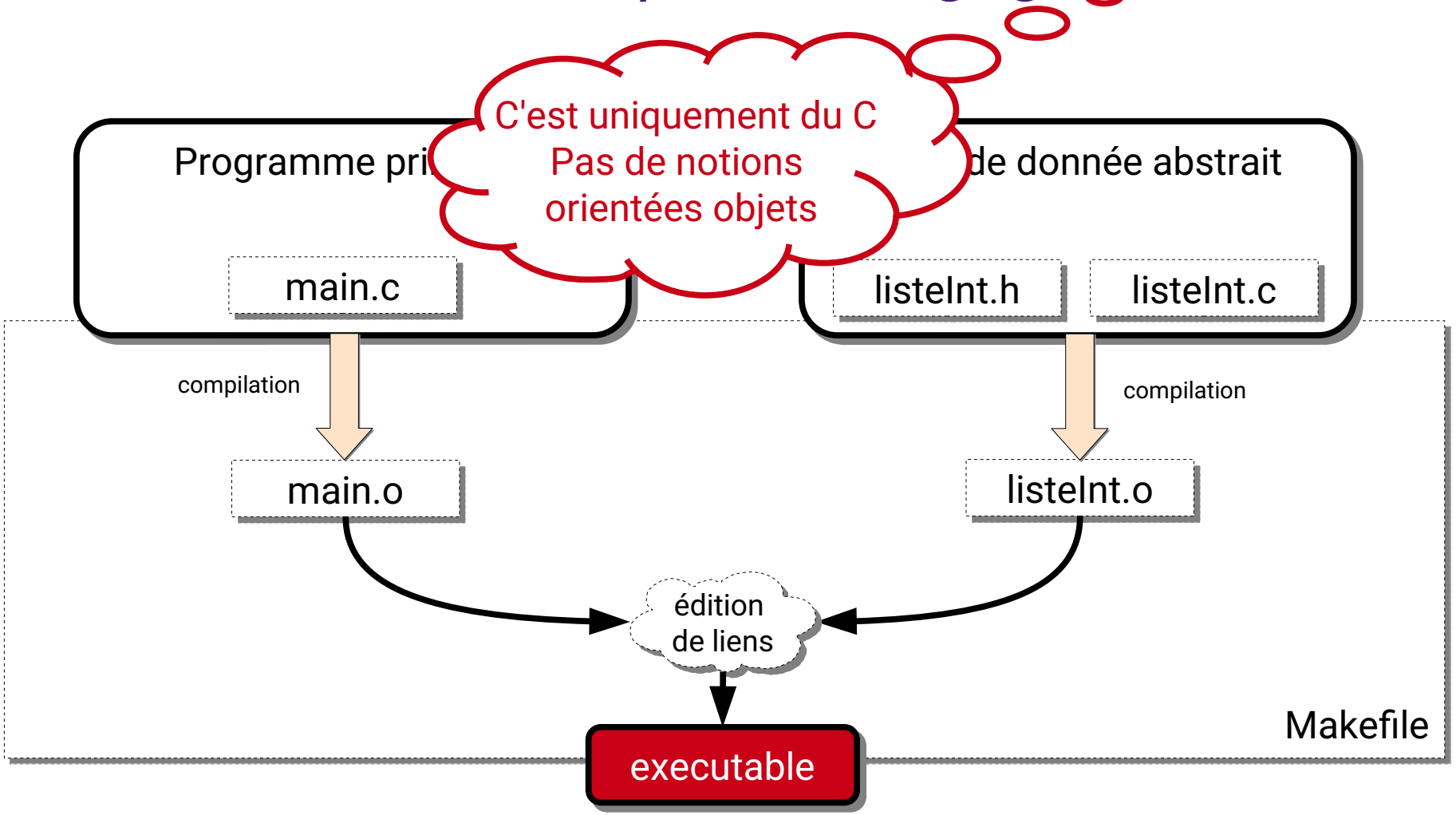


Programming and abstraction
resumé *de l'exemple en langage C*



Programming and abstraction

resumé *de l'exemple en langage C*



Programming and abstraction

Le langage C++

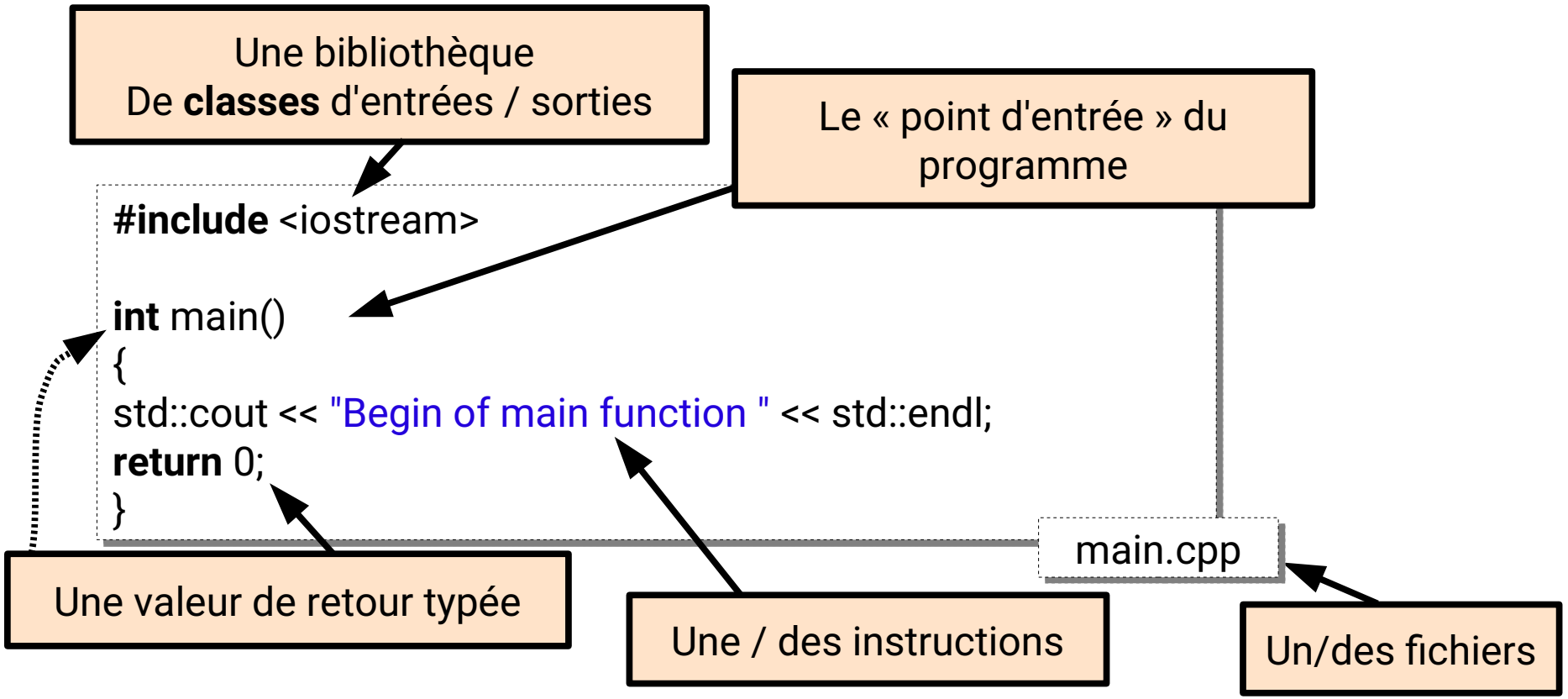
Programming and abstraction

quoi de plus dans le C++ ?

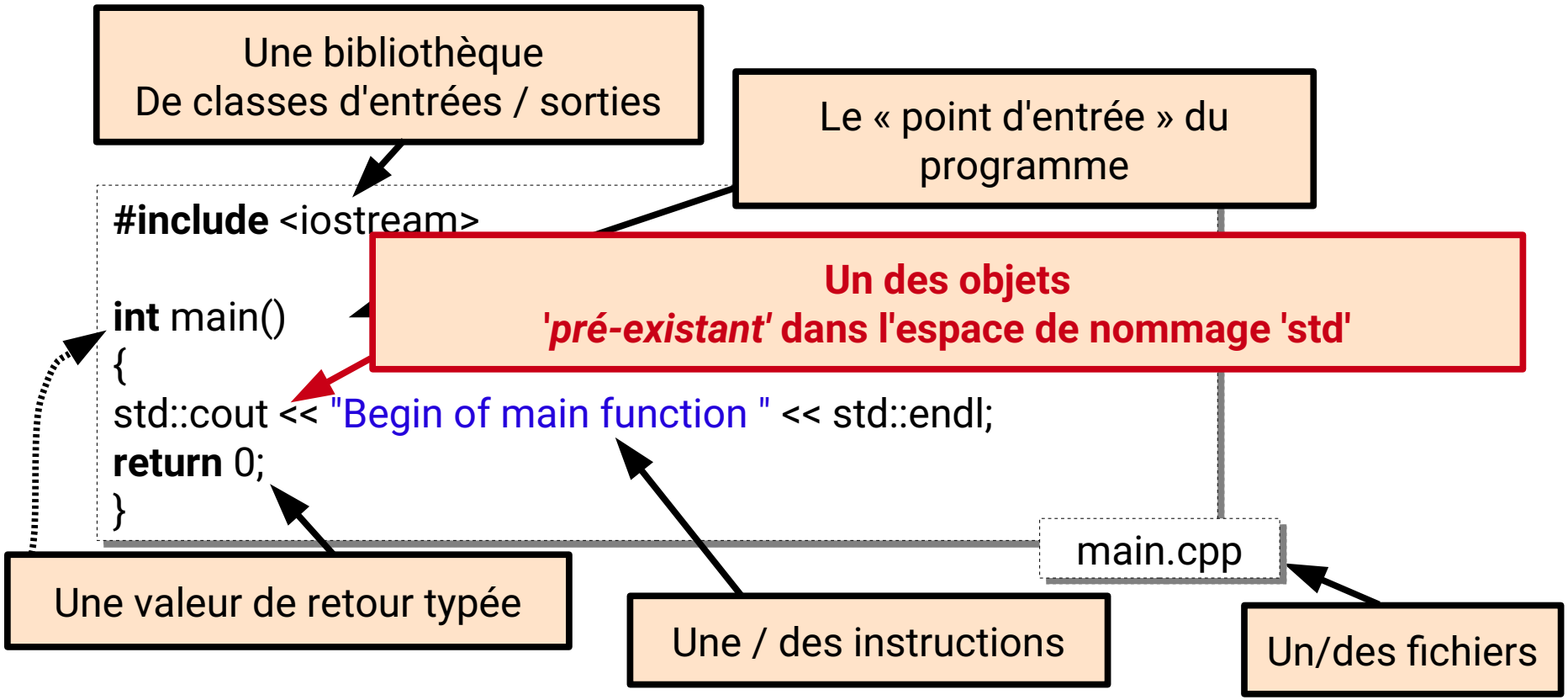
- Un typage fort
  - Surcharge de fonctions, d'opérateurs
- De la généricité : templates…
- La STL : Standard Template Library
  - Containers
  - Algorithms
  - ...
- Toutes les notions de ADT et orientées objets
  - Encapsulation / notion de visibilité
  - Hiérarchie entre les types de données (héritage)
  - Résolution dynamique (à l'exécution) du type effectif d'un objet (typage dynamique, polymorphisme)**, uniquement lorsque nécessaire et demandé par le développeur**
- C++11, 14, 17, 20
  - Lambda expressions, gestion de la concurrence, inférence de type, …

Le langage C++
Procédures et fonctions

Exemple simple en *langage C++*



Exemple simple en *langage C++*



Programming and abstraction

compiler avec un « make » en langage C

```cpp
#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**
      - ≃ faire le diagramme de classe
    - Implémenter la (les) classe(s) → **.cpp**
      - ≃ 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

```cpp
#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

```cpp
#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;
}
```

Inclusion de la bibliothèque associée à la classe

main.cpp

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;
}
```

Inclusion de la bibliothèque associée à la classe

Déclaration d'un Objet [+ initialisation]

main.cpp

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;
```

Inclusion de la bibliothèque associée à la classe

Déclaration d'un Objet [+ initialisation]

Pas de new en C++ !!!

(pour l'instant ;))

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;
}
```

Inclusion de la bibliothèque associée à la classe

Déclaration d'un Objet [+ initialisation]

Affichage de l'objet

main.cpp

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;
}
```

Inclusion de la bibliothèque associée à la classe

Déclaration d'un Objet [+ initialisation]

sérialisation de l'objet sur la sortie standard

main.cpp

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;
}
```

Inclusion de la bibliothèque associée à la classe

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