A <Basic> C++ Course

1 - Introduction

Julien Deantoni

Plan

- Du C++ ? pourquoi ?
- Principales différences avec java ?
  - le langage
  - La mise en oeuvre (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++ ?

- Très utilisé
  - https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018
- Libre
- Performant

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

Pourquoi C++ ?

- Encore très utilisé en finance (surtout combiné au C et en finance)
  - https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018
- Libre et compilation performante

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

Philosophie différente / complémentaire à java : à connaître

Différences avec JAVA

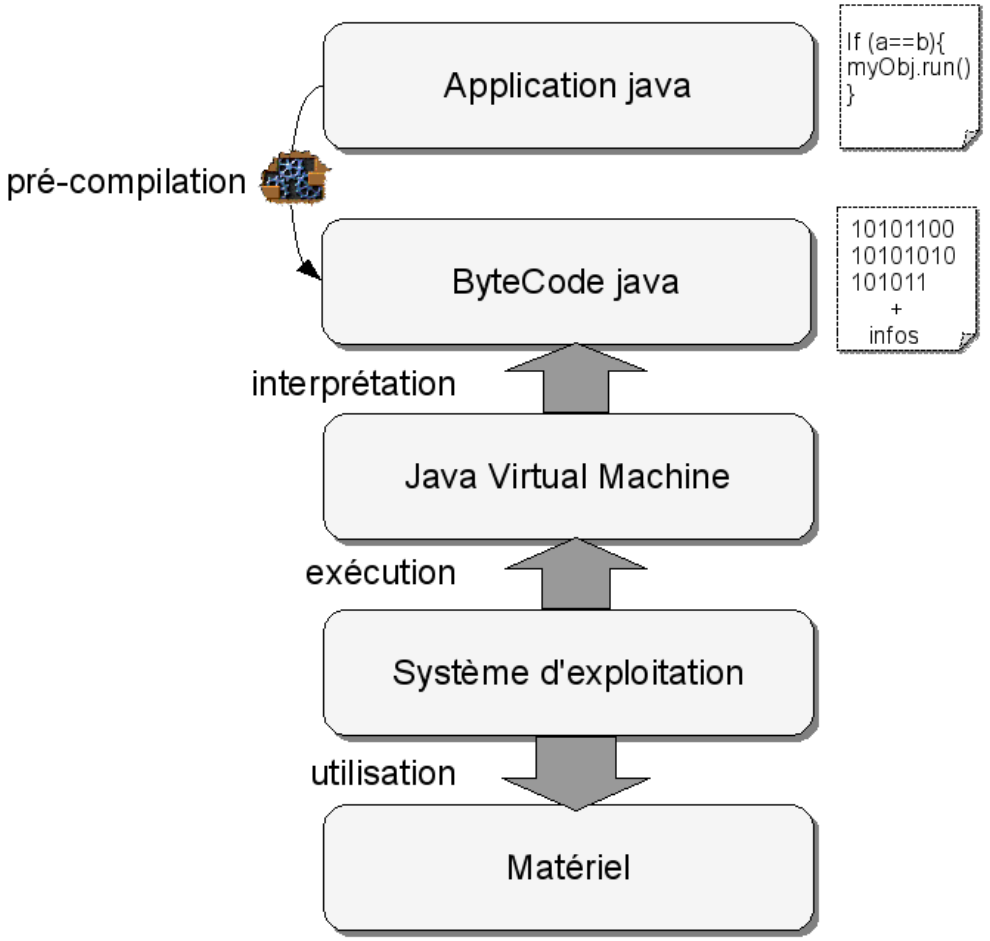
le langage

- 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
  - 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

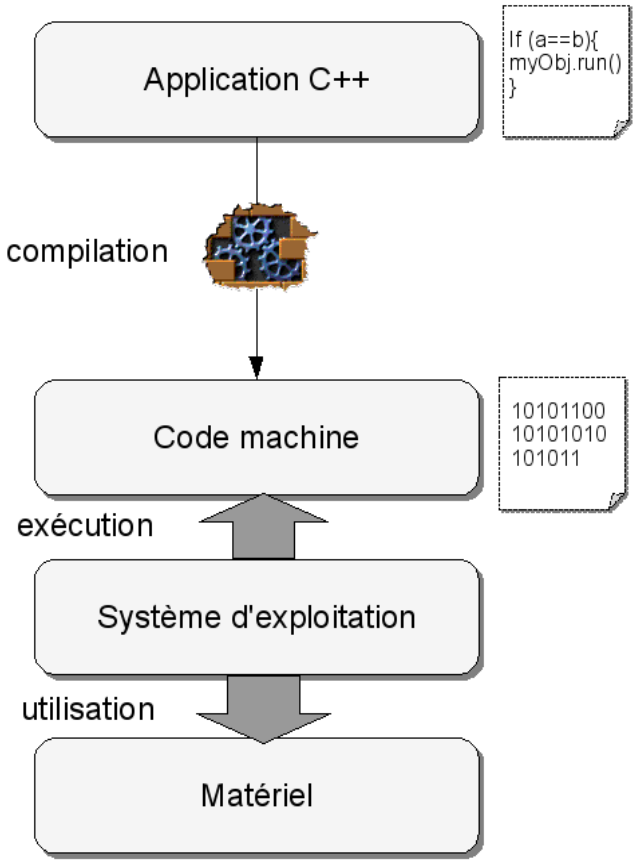


Noter que depuis quelques années, Java et son évolution sont dirigés par Oracle...

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);
```

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.

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++

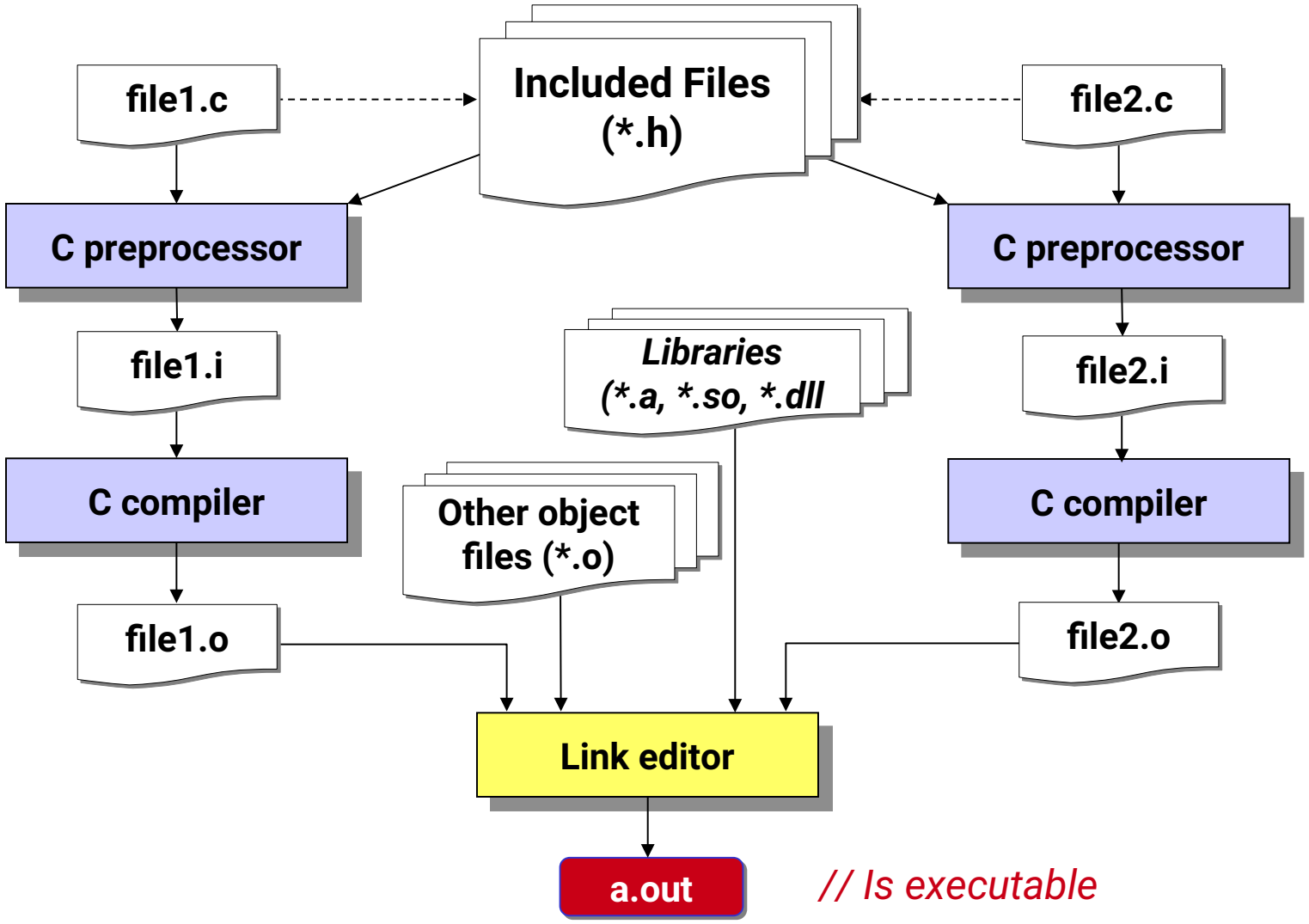
Programming and abstraction

- **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

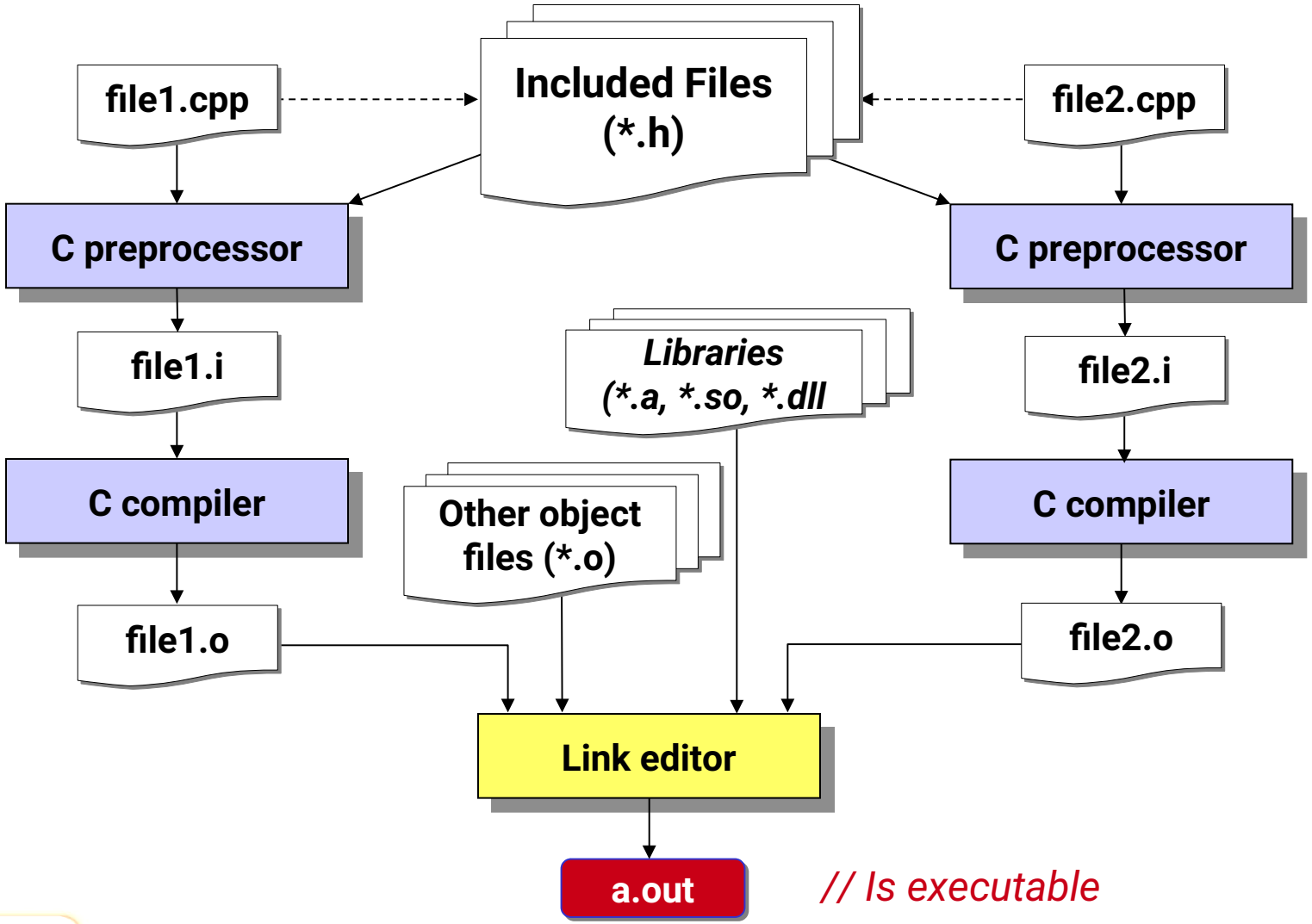
From C to C++
Programming and abstraction

- **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

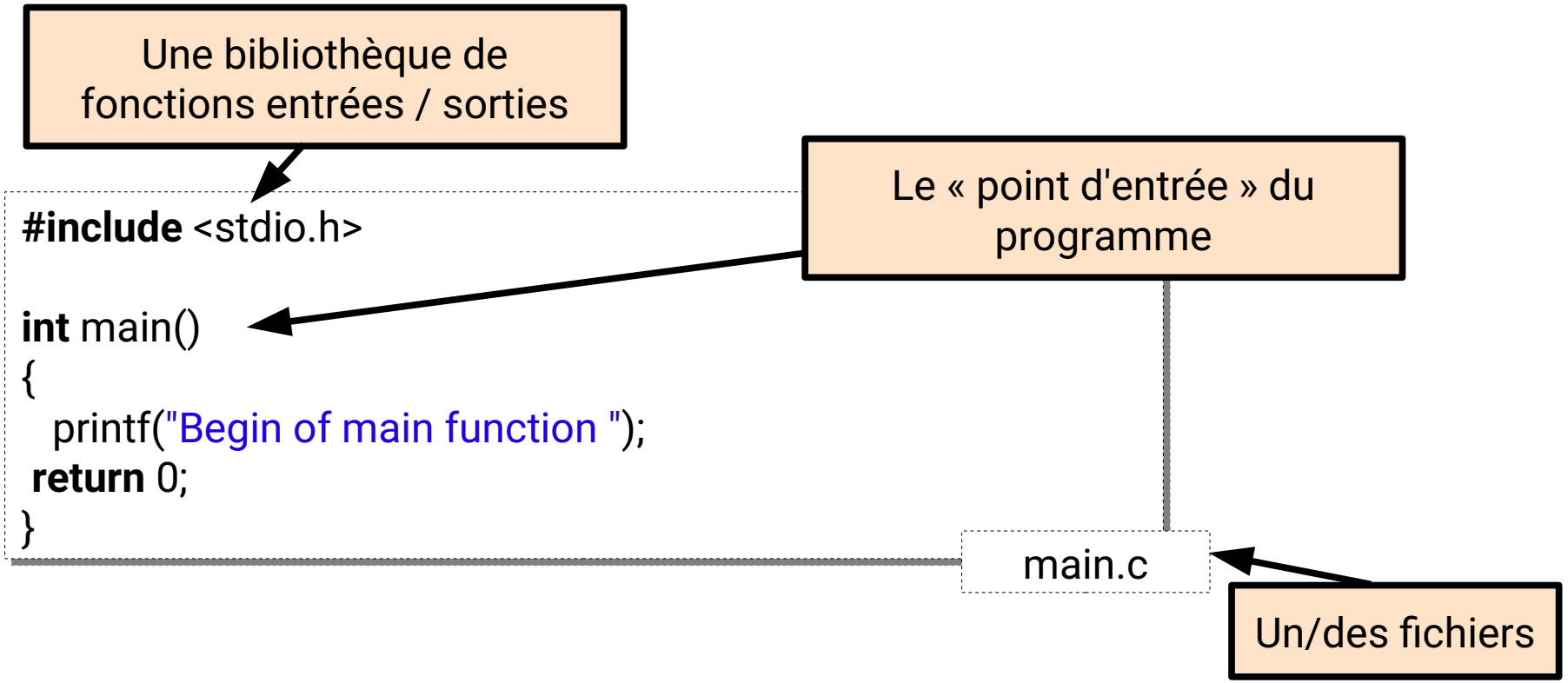
```c
#include <stdio.h>

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

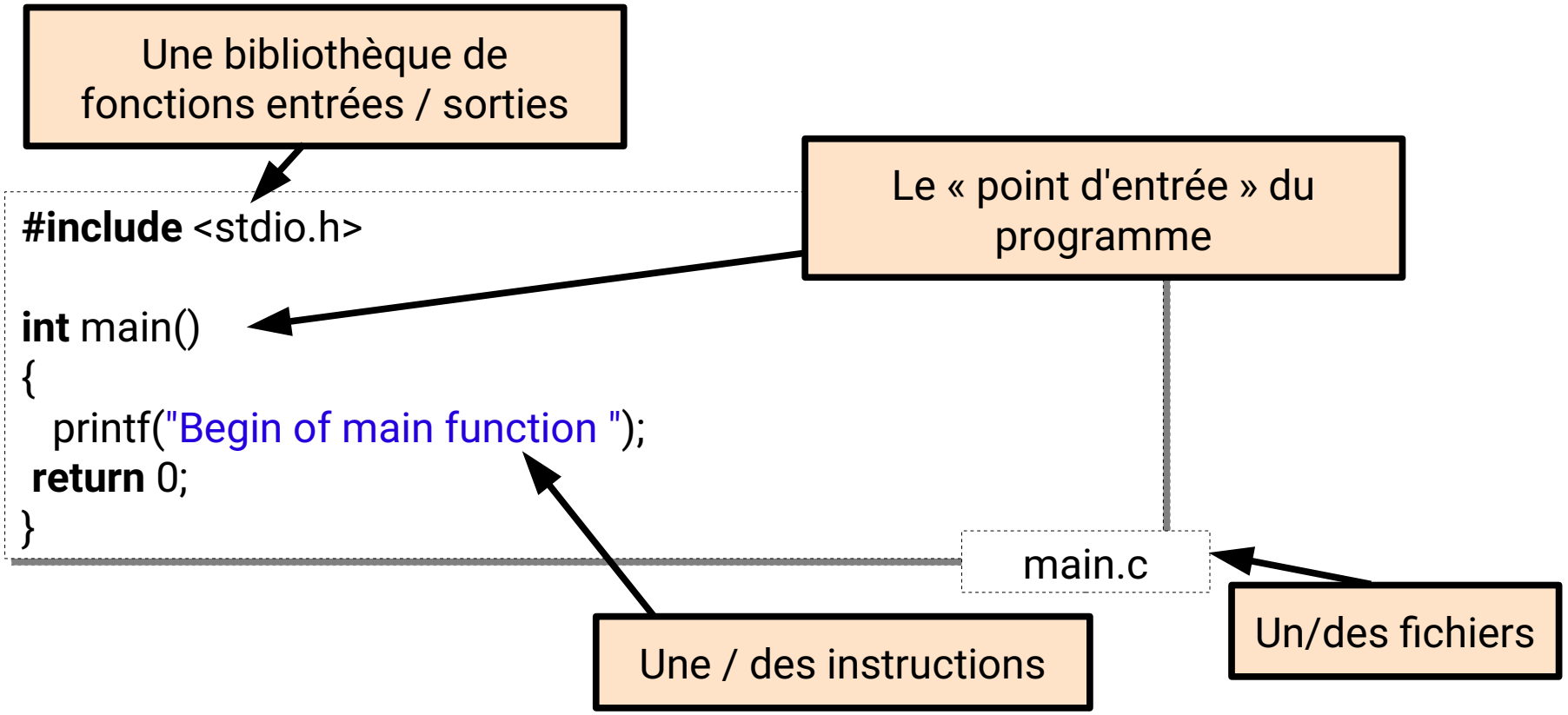
main.c

Un/des fichiers

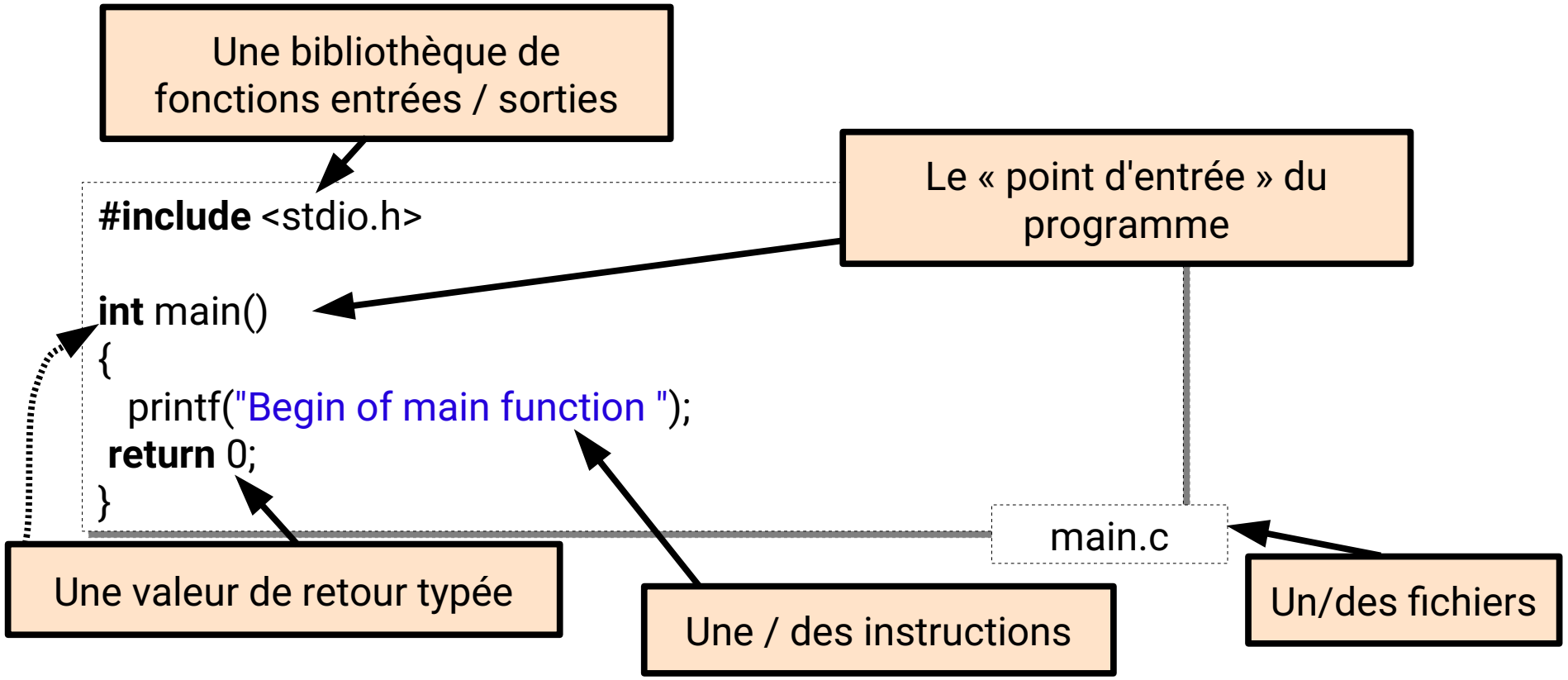
Exemple simple en *langage C*



Exemple simple en *langage C*

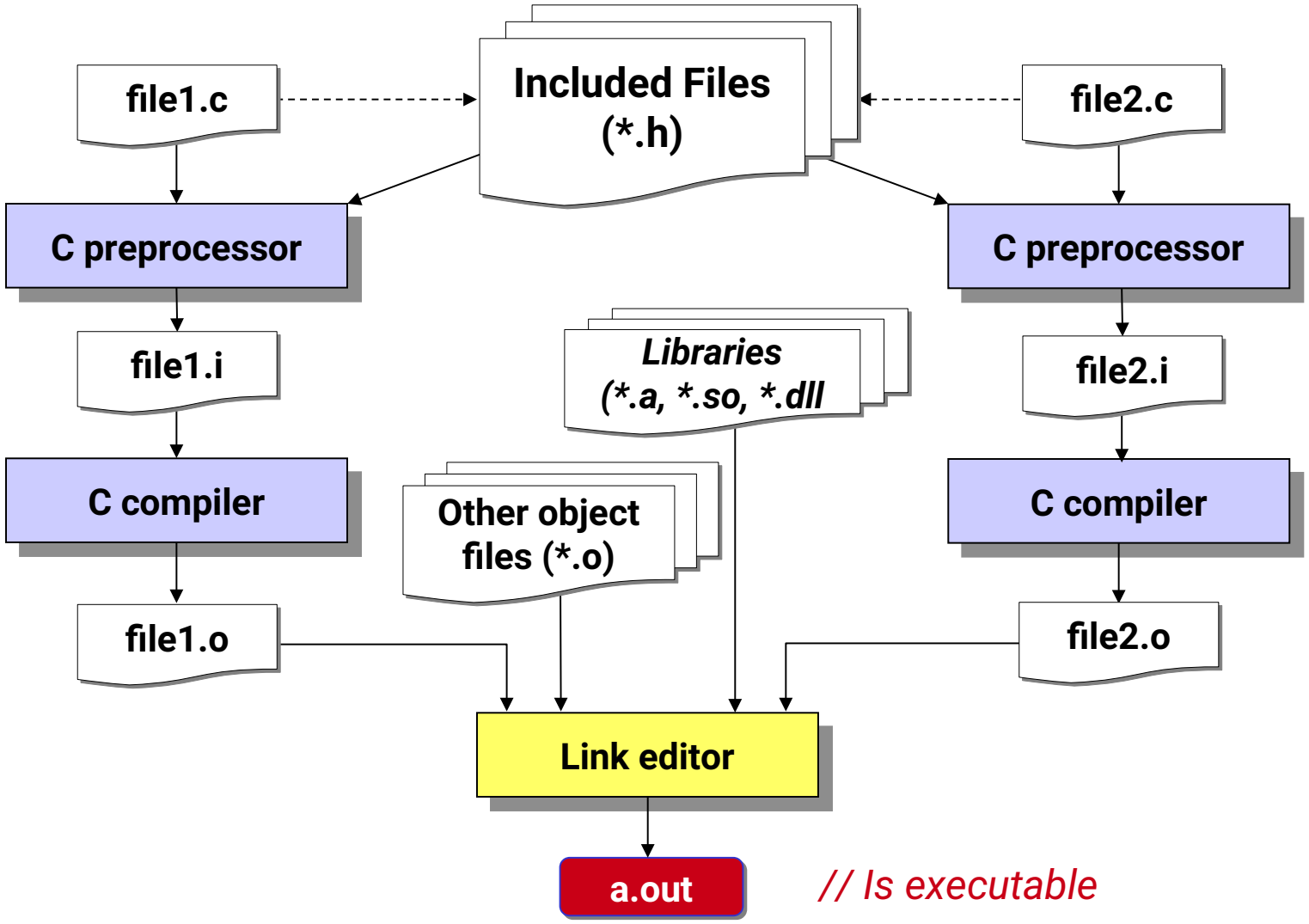


Exemple simple en *langage C*



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

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
```

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)
```

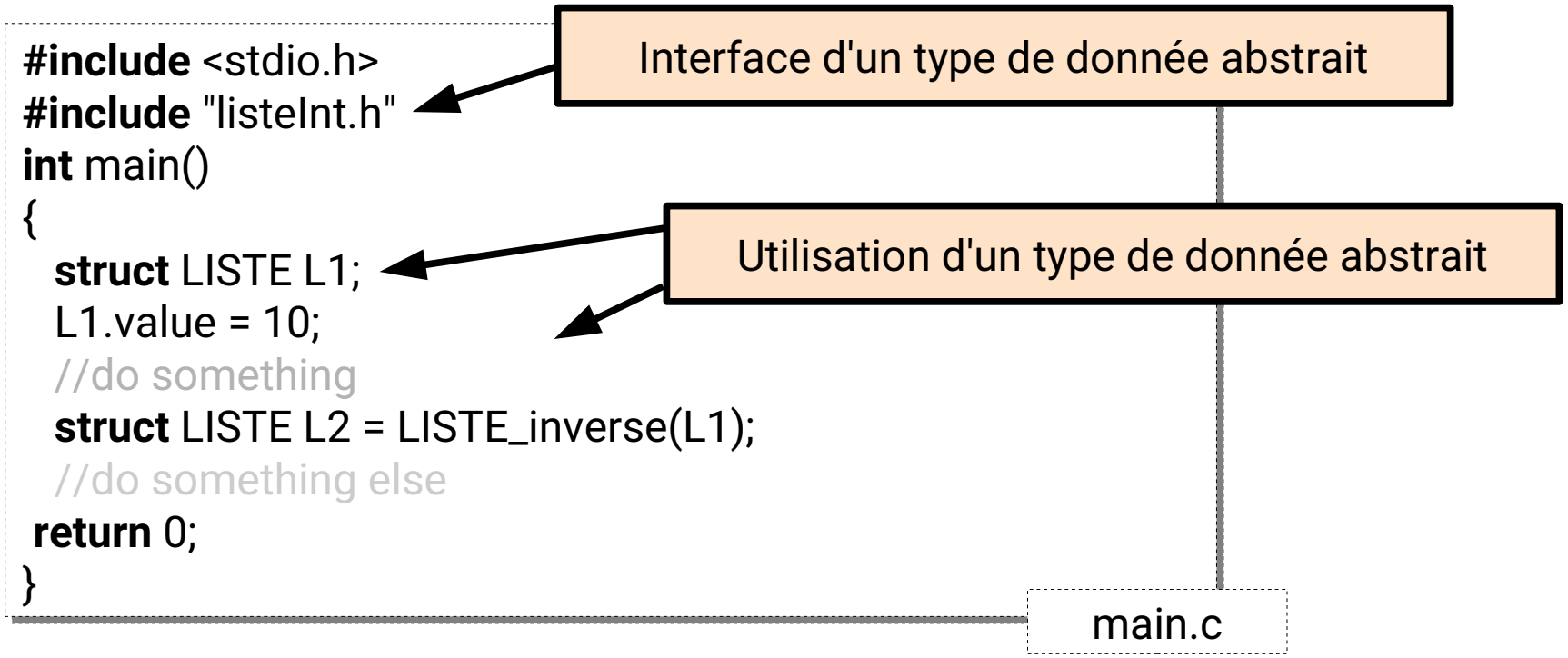
Makefile

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

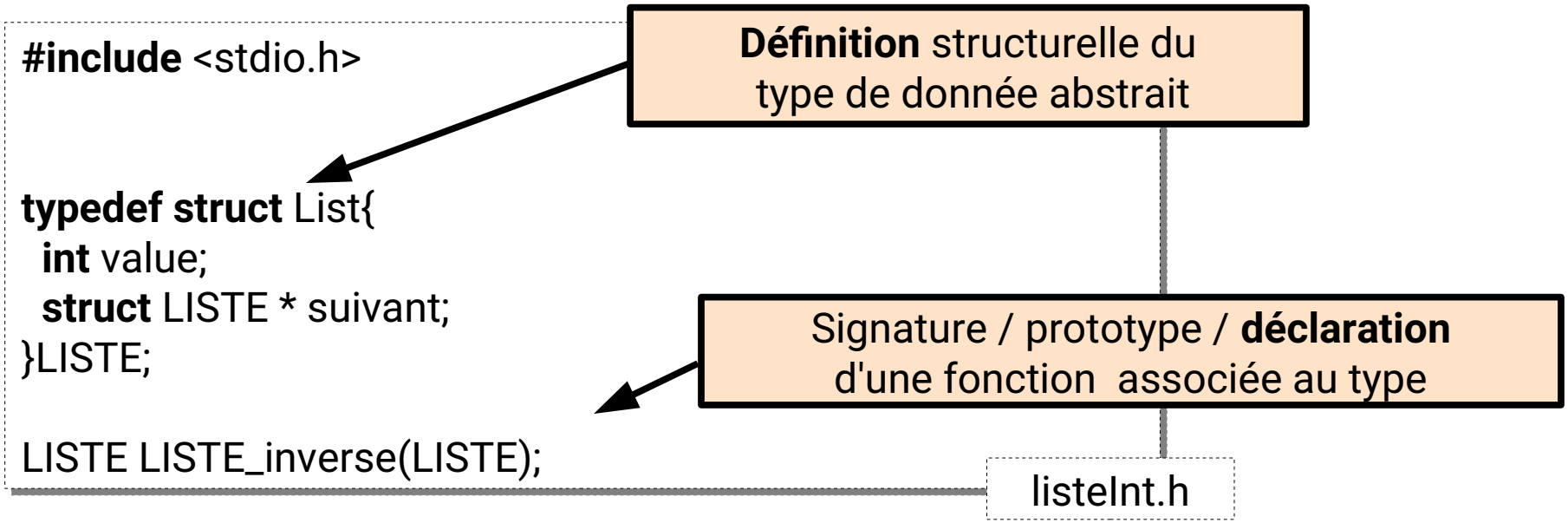
Lancement de la compilation de la cible « clean »

Le langage C
Les types de données abstraits

Exemple en *langage C*



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

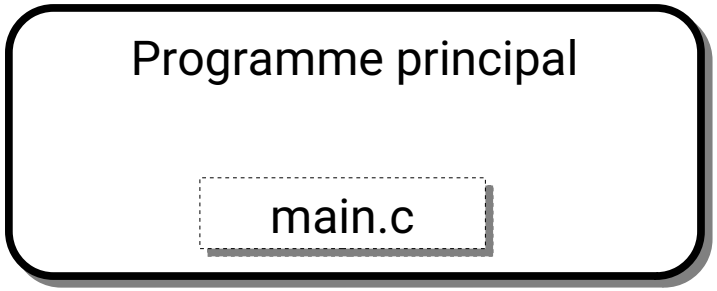
Programming and abstraction

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...)

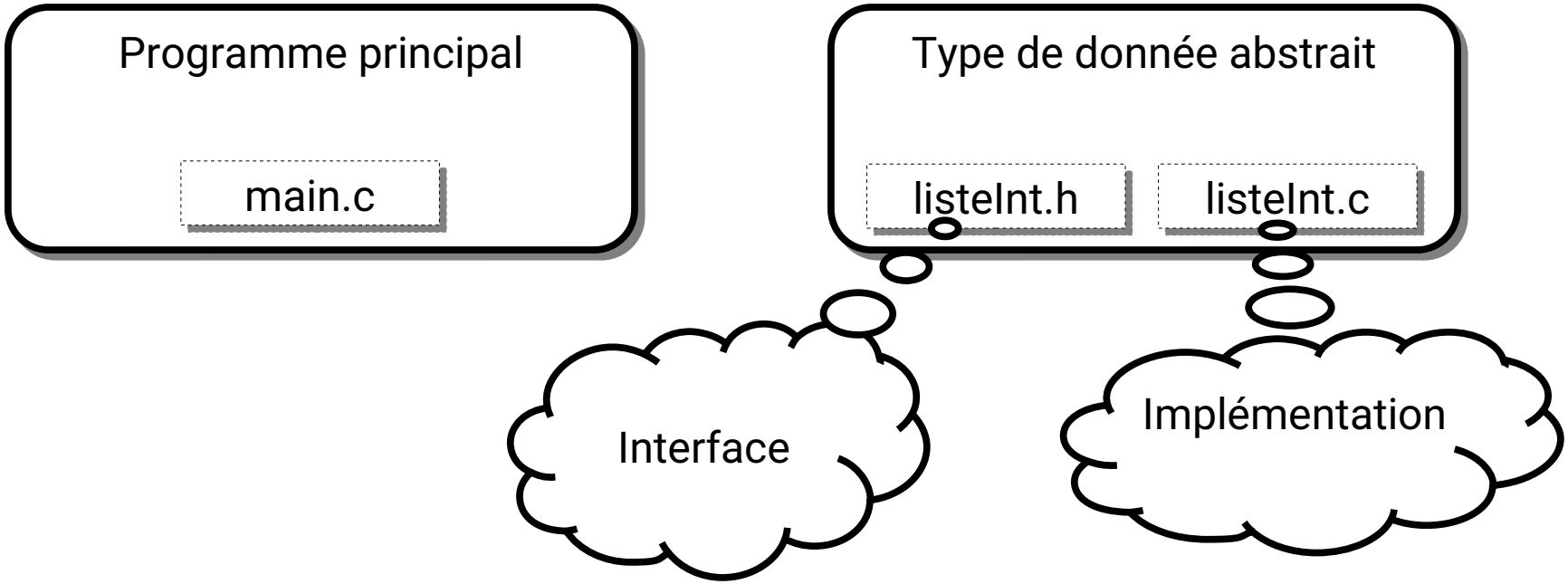
Programming and abstraction

resumé *de l'exemple en langage C*



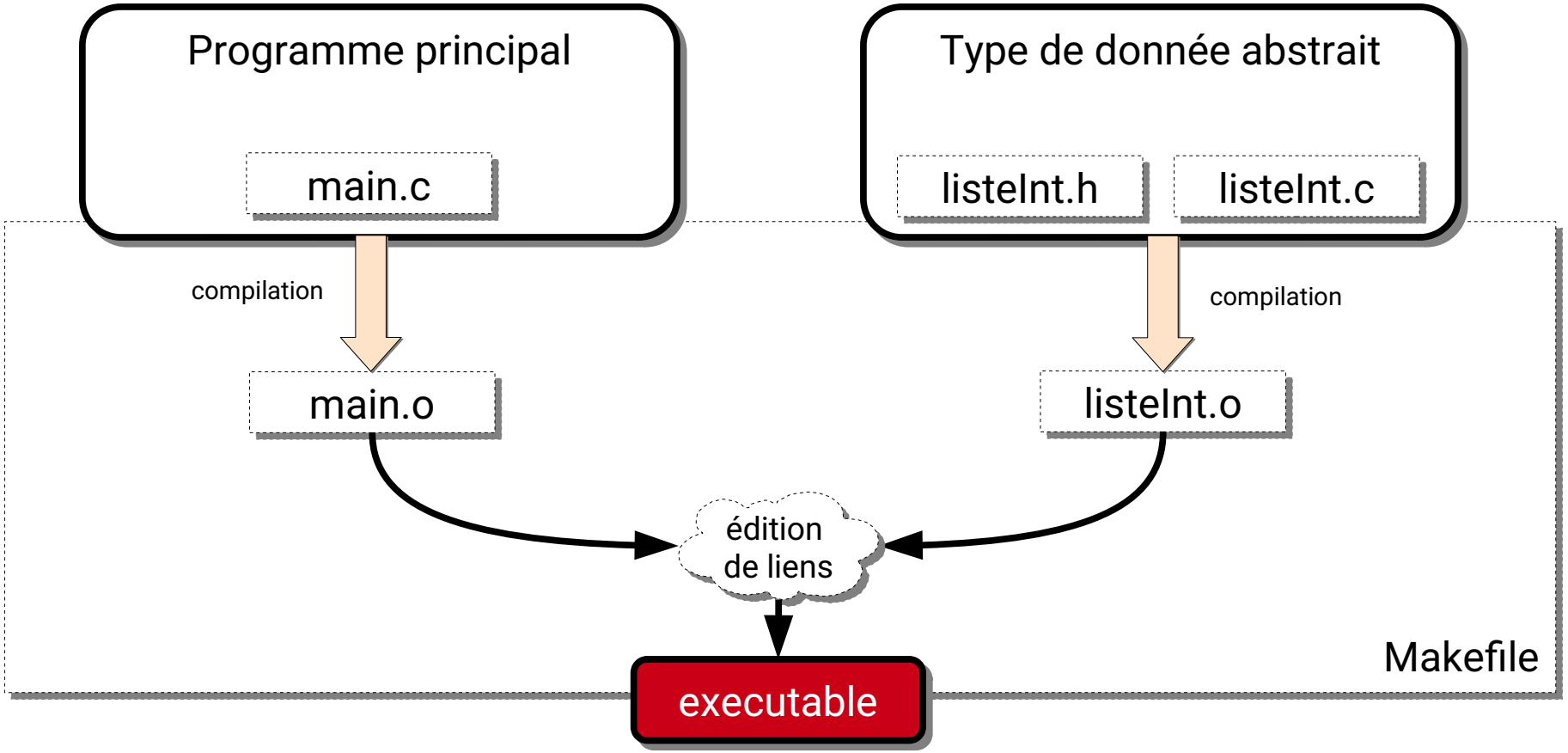
Programming and abstraction

resumé *de l'exemple en langage C*



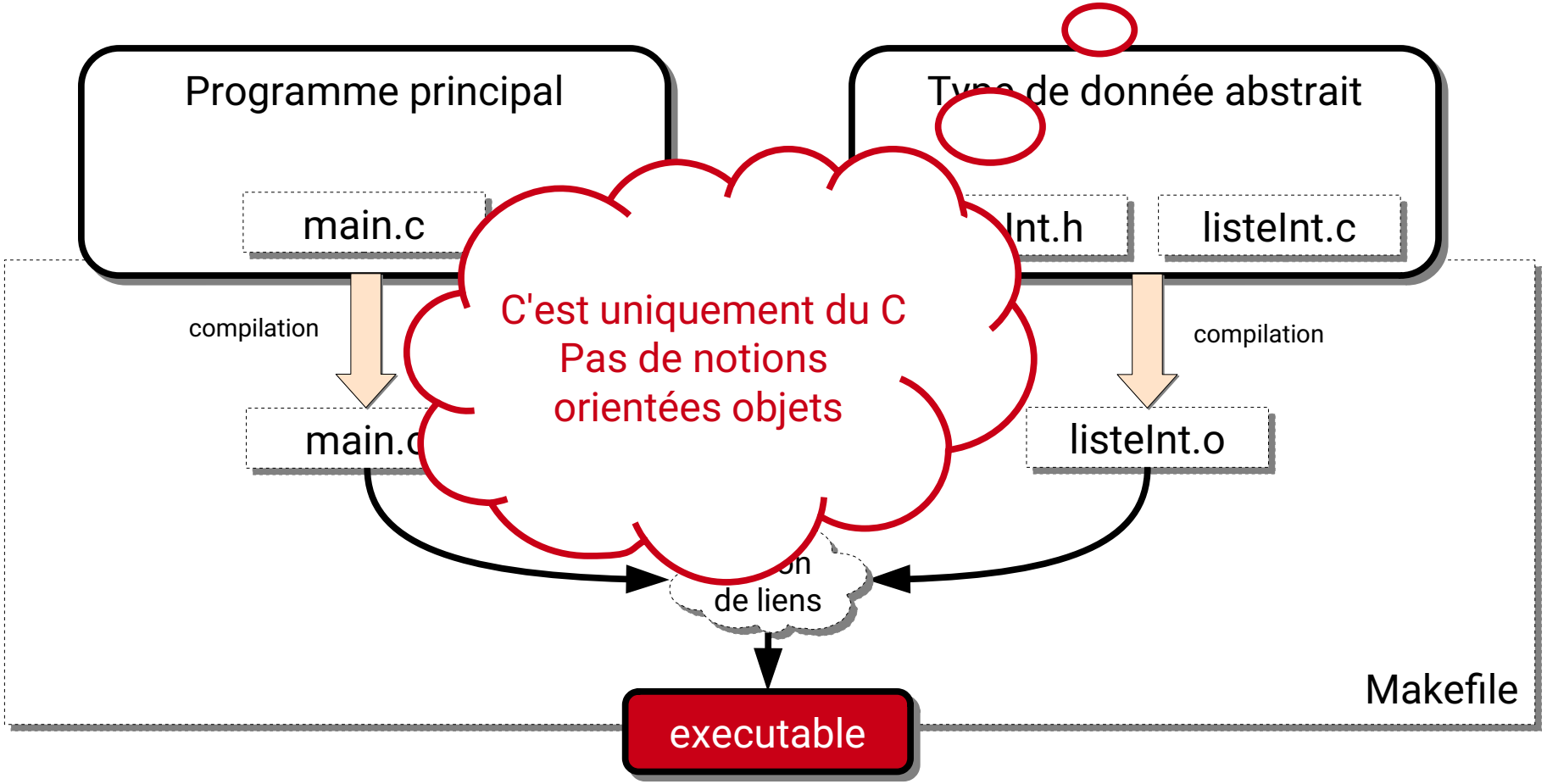
Programming and abstraction

resumé *de l'exemple en langage C*



Programming and abstraction

resumé *de l'exemple en langage C*



Programming and abstraction

Le langage C++

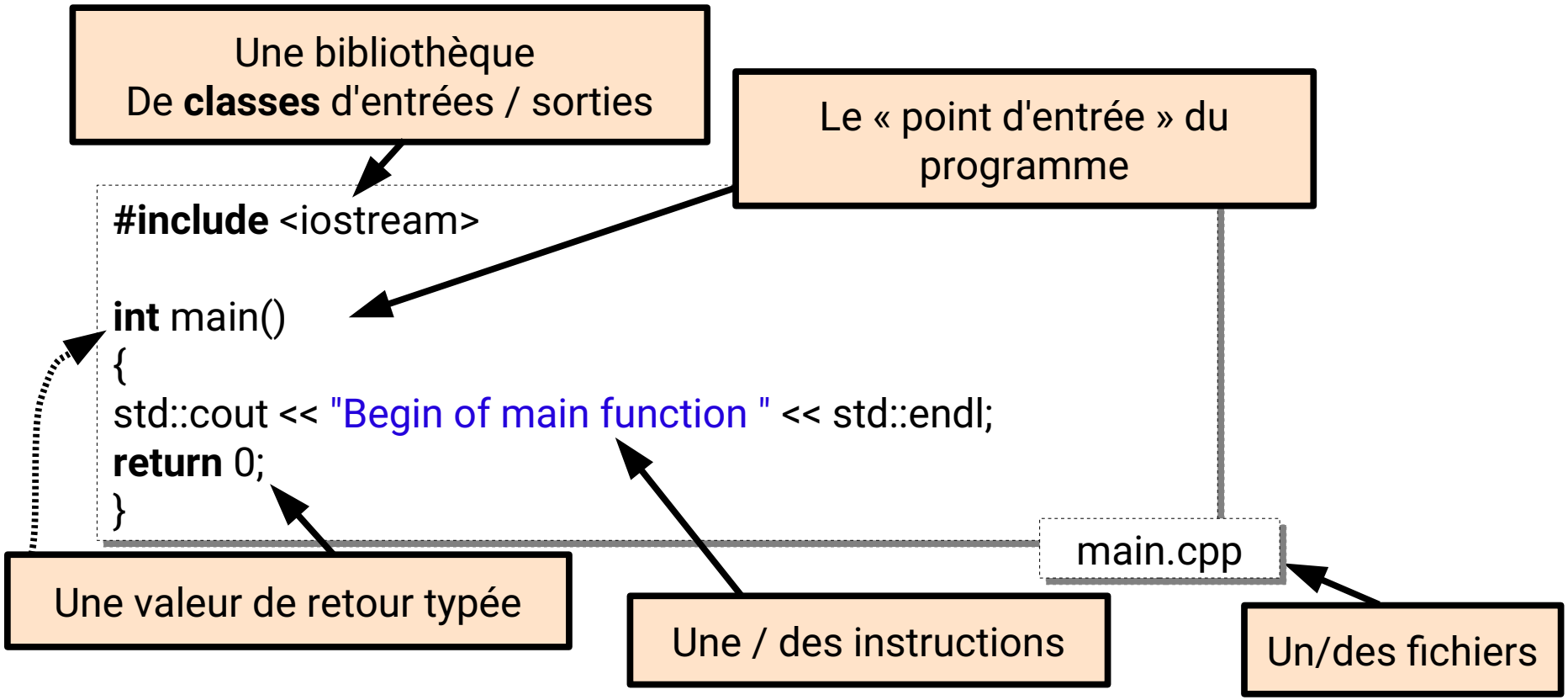
Programming and abstraction

quoi de plus dans le C++ ?

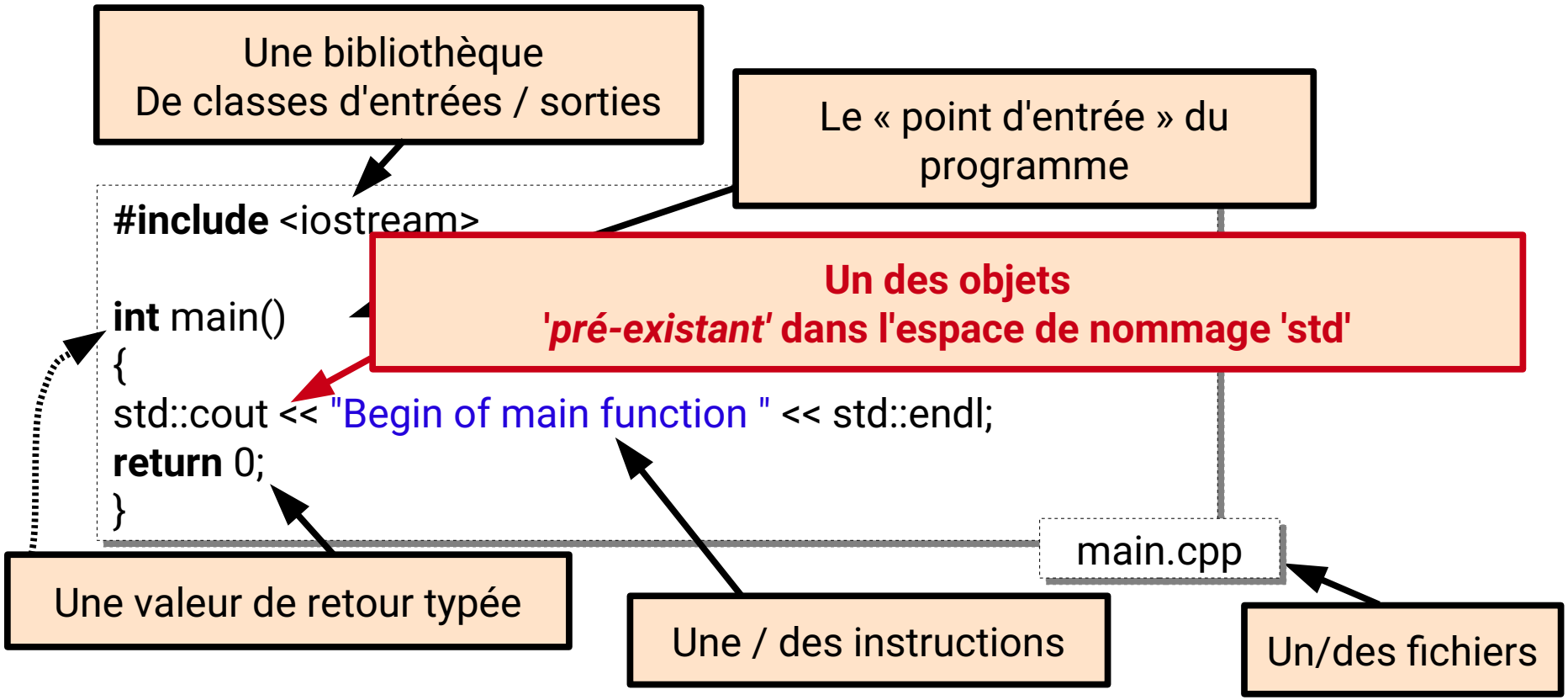
- Un typage fort
  - Surcharge de fonctions, d'opérateurs
- De la généricité : templates…
- La STL : Standard Template Library
  - Containers
  - Algorithms
  - ...
- Toutes les notions orientées objets
  - Encapsulation / notion de visibilité
  - Hiérarchie entre les types de données (héritage)
  -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

Le langage C++
Procédures et fonctions

Exemple simple en *langage C++*



Exemple simple en *langage C++*



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