

# A <Basic> C++ Course

## 2 – namespace et mémoire...

*Julien Deantoni*

# Plan

- Créer ses fonctions utiles
- Espaces de nommage
- Memory concerns
  - statique

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){
    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

```
#include <string>
```

```
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include "myOverkillUtils.h"
```

```
bool isPalindrome(const std::string& s){
    return s==string(s.rbegin(), s.rend());
}
```

myOverkillUtils.cpp

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){
    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

```
#include <string>
```

```
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include "myOverkillUtils.h"
```

```
bool isPalindrome(const std::string& s){
    return s==string(s.rbegin(), s.rend());
}
```

myOverkillUtils.cpp



## Complexity of isPalindrome

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

```
#include <string>
```

```
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include "myUtils.h"
```

```
bool isPalindrome(const std::string& s){
    return s==string(s.rbegin(), s.rend());
}
```

myOverkillUtils.cpp

# sketchy makefile example

EXE\_NAME=executable

LINK\_CXX=g++ -std=c++11

COMPILE\_CXX=g++ -c -Wall -Wextra -std=c++11

example: main.o **utils.o**

\$(LINK\_CXX) main.o **utils.o** -o \$(EXE\_NAME)

main.o: main.cpp

\$(CXX) main.cpp

**utils.o: myOverkillUtils.cpp myOverkillUtils.h**

**\$(CXX) myOverkillUtils.cpp**

Makefile

```
jdeanton@linux-hani: make
g++ -g -c main_palindrome.cpp -o main_palindrome.o [options]
g++ -g -c myUtils.cpp -o myUtils.o [options]
g++ main_palindrome.o myUtils.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
```

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

```
#include <string>

bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include "myUtils.h"
```

myUtils.cpp



Ce code ne marche pas si des définitions sont données dans le .h

```
# sketchy
EXE_NAME=executable
LINK_CXX=g++ -std=c++11
COMPILE_CXX=g++ -c -Wall -Wextra -std=c++11
```

```
example: main.o utils.o
$(LINK_CXX) main.o utils.o -o $(EXE_NAME)
main.o: main.cpp
$(CXX) main.cpp
Utils.o: myOverkillUtils.cpp myOverkillUtils.h
$(CXX) myOverkillUtils.cpp
```

Makefile

```
jdeanton@linux-hani: make -f Makefile_v1
g++ -g -c main_palindrome.cpp -o main_palindrome.o [options]
g++ -g -c myUtils.cpp -o myUtils.o [options]
g++ main_palindrome.o myUtils.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
```

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
#include "someStuff.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);
    std::cout << std::boolalpha << res<<std::endl;
    return EXIT_SUCCESS;
}
```

main.cpp

```
jdeanton@linux-hani: make
g++ -g -c main_palindrome.cpp -o main_palindrome.o [options]
g++ -g -c myUtils.cpp -o myUtils.o [options]
g++ -g -c someStuff.cpp -o someStuff.o [options]
g++ main_palindrome.o myUtils.o someStuff.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
```

```
#include <string>

struct Foo{
    //[...]
};
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include <string>

void stuff1();
```

someStuff.h



# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
#include "someStuff.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);
    std::cout << std::boolalpha << res<<std::endl;
    return EXIT_SUCCESS;
}
```

main.cpp

```
jdeanton@linux-hani: make
g++ -g -c someStuff.cpp -o someStuff.o [options]
g++ main_palindrome.o myUtils.o someStuff.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
```

```
#include <string>

struct Foo{
    //[...]
};
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include <string>
#include "myOverkillUtils.h"

void stuff1(Foo f);
```

someStuff.h

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
#include "someStuff.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);
    std::cout << std::boolalpha << res<<std::endl;
    return EXIT_SUCCESS;
}
```

main.cpp

```
jdeanton@linux-hani: make
g++ -g -c someStuff.cpp -o someStuff.o [options]
g++ main_palindrome.o myUtils.o someStuff.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
```

```
#include <string>

struct Foo{
    //[...]
};
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include <string>
#include "myOverkillUtils.h"

void stuff1(Foo f);
```

someStuff.h

## So What ??? It works !

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
#include "someStuff.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);
    std::cout << std::boolalpha << res<<std::endl;
    return EXIT_SUCCESS;
}
```

main.cpp

```
#include <string>

struct Foo{
    //[...]
};
bool isPalindrome(const std::string& s);
```

myOverkillUtils.h

```
#include <string>
#include "myOverkillUtils.h"

void stuff1(Foo f);
```

someStuff.h

```
jdeanton@linux-hani: make
g++ -g -c someStuff.cpp -o someStuff.o [options]
g++ main_palindrome.o myUtils.o someStuff.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
jdeanton@linux-hani: make clean
rm -f main_palindrome.o myUtils.o someStuff.o
jdeanton@linux-hani: make
g++ -g -c main_palindrome.cpp -o main_palindrome.o [options]
In file included from someStuff.h:1:0,
      from main_palindrome.cpp:4:
myUtils.h:8:8: error: redefinition of 'struct Foo'
struct Foo{
      ^
In file included from main_palindrome.cpp:3:0:
myUtils.h:8:8: error: previous definition of 'struct Foo'
struct Foo{
      ^
Makefile:41: recipe for target 'main_palindrome.o' failed
make: *** [main_palindrome.o] Error 1
```

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
#include "someStuff.h"

int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);
    std::cout << std::boolalpha << res<<std::endl;
    return EXIT_SUCCESS;
}
```

main.cpp

```
#ifndef _MYUTILS_H_
#define _MYUTILS_H_

#include <string>
struct Foo{
};
bool isPalindrome(const std::string& s);

#endif
```

myOverkillUtils.h

```
#ifndef _SOMESTUFF_H_
#define _SOMESTUFF_H_

#include <string>
#include "myOverkillUtils.h"

void stuff1(Foo f);

#endif
```

someStuff.h

```
jdeanton@linux-hani: make
g++ -g -c someStuff.cpp -o someStuff.o [options]
g++ main_palindrome.o myUtils.o someStuff.o [options]
jdeanton@linux-hani: ./palindrome.exe
true
jdeanton@linux-hani: make clean
rm -f main_palindrome.o myUtils.o someStuff.o
jdeanton@linux-hani: make
g++ -g -c main_palindrome.cpp -o main_palindrome.o [options]
In file included from someStuff.h:1:0,
      from main_palindrome.cpp:4:
myUtils.h:8:8: error: redefinition of 'struct Foo'
struct Foo{
^
In file included from main_palindrome.cpp:3:0:
myUtils.h:8:8: error: previous definition of 'struct Foo'
struct Foo{
^
Makefile_v1:41: recipe for target 'main_palindrome.o' failed
make: *** [main_palindrome.o] Error 1
```



Protéger systématiquement le contenu d'un .h par une directive pre processeur et ainsi éviter l'inclusion multiple.

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

```
#ifndef _MYUTILS_H_
#define _MYUTILS_H_
#include <string>

bool isPalindrome(const std::string& s); //visible

#endif
```

myOverkillUtils.h

```
#include "myOverkillUtils.h"
#include <iomanip>
#include <ctime>
#include <fstream>

#define LOGFILE "log.txt"

void logCall(const std::string& s){ //invisible
depuis l'extérieur !
    std::ofstream fout;
    fout.open(LOGFILE, std::ios::app);

    std::time_t t = std::time(nullptr);
    std::tm tm = *std::localtime(&t);

    fout << std::put_time(&tm, "%F %T");
    fout << " : " << s << std::endl;
    fout.close();
}

bool isPalindrome(const std::string& s){
    logCall("isPalindrome");
    return s==std::string(s.rbegin(), s.rend());
}
```

myOverkillUtils.cpp

# Créer ses fonctions utiles

```
#include <iostream>
#include <string>
#include "myOverkillUtils.h"
```

```
int main(){

    std::string s = "repiper";
    bool res = isPalindrome(s);

    std::cout << std::boolalpha << res<<std::endl;

    return EXIT_SUCCESS;
}
```

main.cpp

```
#ifndef _MYUTILS_H_
#define _MYUTILS_H_
#include <string>

bool isPalindrome(const std::string& s); //visible

#endif
```

myOverkillUtils.h

```
#include "myUtils.h"
#include <iomanip>
#include <ctime>
#include <fstream>

#define LOGFILE "log.txt"
namespace{
    void logCall(const std::string& s){ //VRAIMENT
        invisible depuis l'extérieur !
        std::ofstream fout;
        fout.open(LOGFILE, std::ios::app);

        std::time_t t = std::time(nullptr);
        std::tm tm = *std::localtime(&t);

        fout << std::put_time(&tm, "%F %T");
        fout << " : " << s << std::endl;
        fout.close();
    }
}

bool isPalindrome(const std::string& s){
    logCall("isPalindrome");
    return s==std::string(s.rbegin(), s.rend());
}
```

myOverkillUtils.cpp

# Plan

- Créer ses fonctions utiles
  - Ajout de cible dans le Makefile
  - Attention aux inclusions multiples
- **Espaces de nommage**
- Memory concerns
  - statique

# Espace de nommage

## principes

- Mécanisme de remplacement partiel d'un *package*
  - Imbrication possible
  - Définition possible dans plusieurs fichiers
  - Pas de lien avec la notion de visibilité (sauf anonyme)
    - soient deux classes **A** et **B**. **A** a les mêmes privilèges d'accès aux membres de **B** qu'ils soient dans le même *package* ou pas; et réciproquement.



# Espace de nommage

## mise en œuvre

- Déclaration / Définition

```
namespace itsName;    //déclaration
```

```
namespace itsName      //définition
{
    class A
    {
        // définition de la classe A
    };

    const double PI=3.1415927;

    class B
    {
        // définition de la classe B
    };
}
```

# Espace de nommage

## mise en œuvre

- Accès aux membres
  - Au sein d'un même espace de nommage
    - Rien de particulier

```
namespace itsName           //définition
{
    class A
    {
        // définition de la classe A
    };

    const double PI=3.1415927;

    class B
    {
        A myA;
        ...
    };
}
```

- En dehors de l'espace de nommage
  - Nom qualifié
  - La clause *using namespace*

# Espace de nommage

## mise en œuvre

- Accès aux membres
  - Au sein d'un même espace de nommage
    - Rien de particulier
  - En dehors de l'espace de nommage
    - Nom qualifié

```
namespace itsName           //définition
{
    class A
    {
        // définition de la classe A
    };

    const double PI=3.1415927;
}
class B
{
    itsName::A myA;
    ...
}
```

# Espace de nommage

## mise en œuvre

- Accès aux membres
  - Au sein d'un même espace de nommage
    - Rien de particulier
  - En dehors de l'espace de nommage
    - Nom qualifié

```
namespace itsName           //définition 1
{
    namespace anotherName //définition 2
    {
        class A
        {
            // définition de la classe A
        };
    }
}

class B
{
    itsName::anotherName::A myA;
    ...
}
```

- La clause *using namespace*

# Espace de nommage

## mise en œuvre

- Accès aux membres
  - Au sein d'un même espace de nommage
    - Rien de particulier
  - En dehors de l'espace de nommage
    - Nom qualifié
    - La clause *using namespace*



```
namespace itsName           //définition
{
    class A
    {
        // définition de la classe A
    };
}

using namespace itsName;
class B
{
    A myA;
    ...
}
```

# Espace de nommage

## mise en œuvre

- Accès aux membres
  - Au sein d'un même espace de nommage
    - Rien de particulier
  - En dehors de l'espace de nommage
    - Nom qualifié
    - La clause *using namespace*



```
namespace itsName           //définition
{
    class A
    {
        // définition de la classe A
    };
    const double PI=3.1415927;
}
using itsName::A;
class B
{
    A myA;
    double myPI = itsName::PI;
}
```

# Plan

- Résumé
- Intro (fin)
  - Création de classe en C++
- Espaces de nommage
- **Memory concerns**
  - **Reservation statique**

# Memory concerns...

- Il existe deux manière de créer des objets en mémoire : statique et dynamique.
- Nous allons d'abord détailler la création statique d'objet.
- Les représentations qui suivent ne sont pas contractuelles mais abstraient la manière dont la mémoire est gérée.
- Ceci permet de rappeler la syntaxe (assez perturbante au début) liée à la manipulation des entités telles que les pointeurs et/ou les références



# Memory concerns...

- Les variables / objets sont stockés en mémoire

```
main(){  
  
}
```



Memory

# Memory concerns...

- Les variables / objets sont stockés en mémoire

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

*i*:int

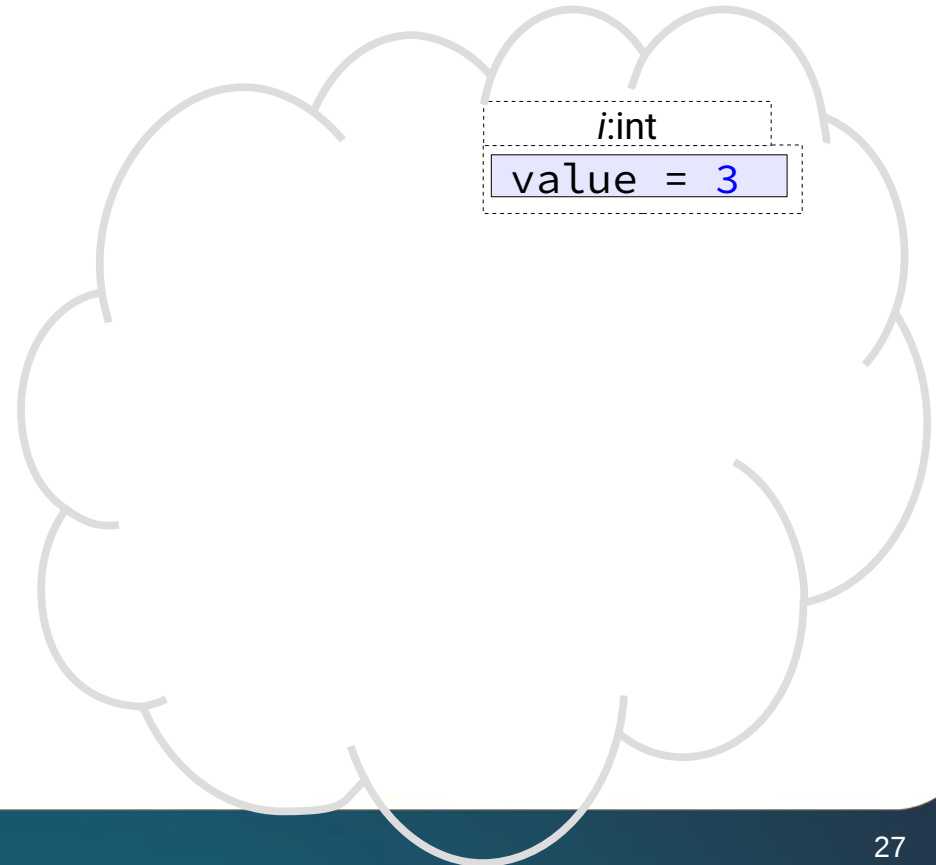
value = ??

Memory

# Memory concerns...

- Les variables / objets sont stockés en mémoire

```
main(){  
  int i;  
  i=3;  
}
```



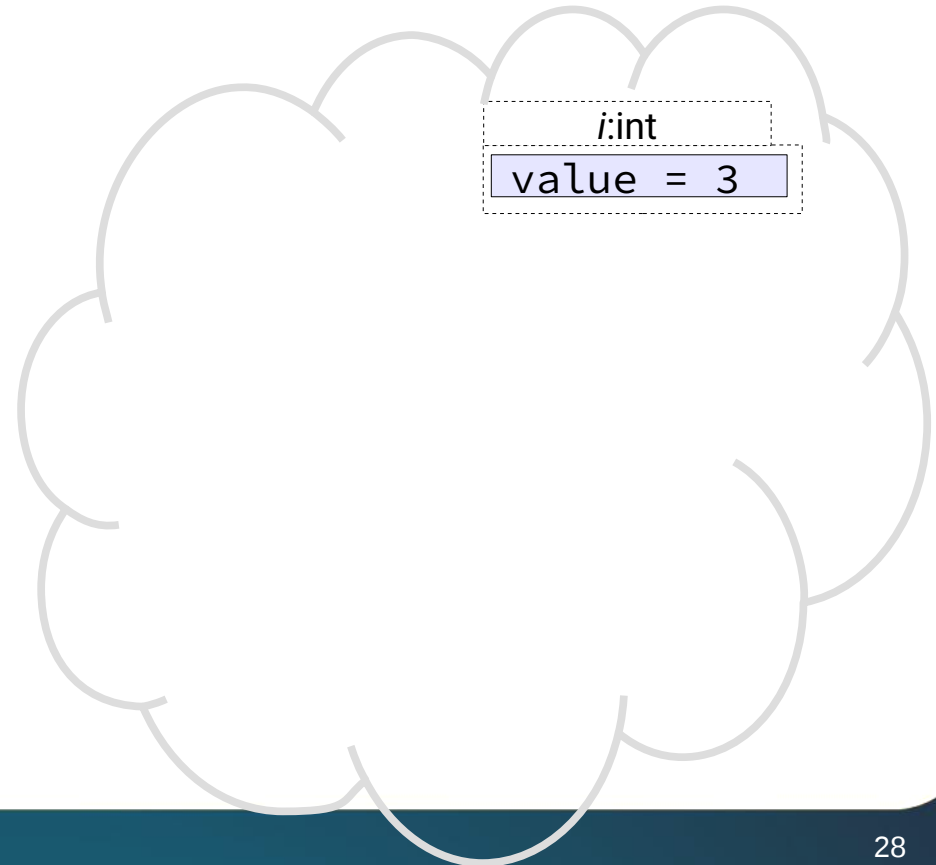
# Memory concerns...

## *passage par copie / valeur*

- Spécifié dans la déclaration & la définition d'une fonction

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
}
```



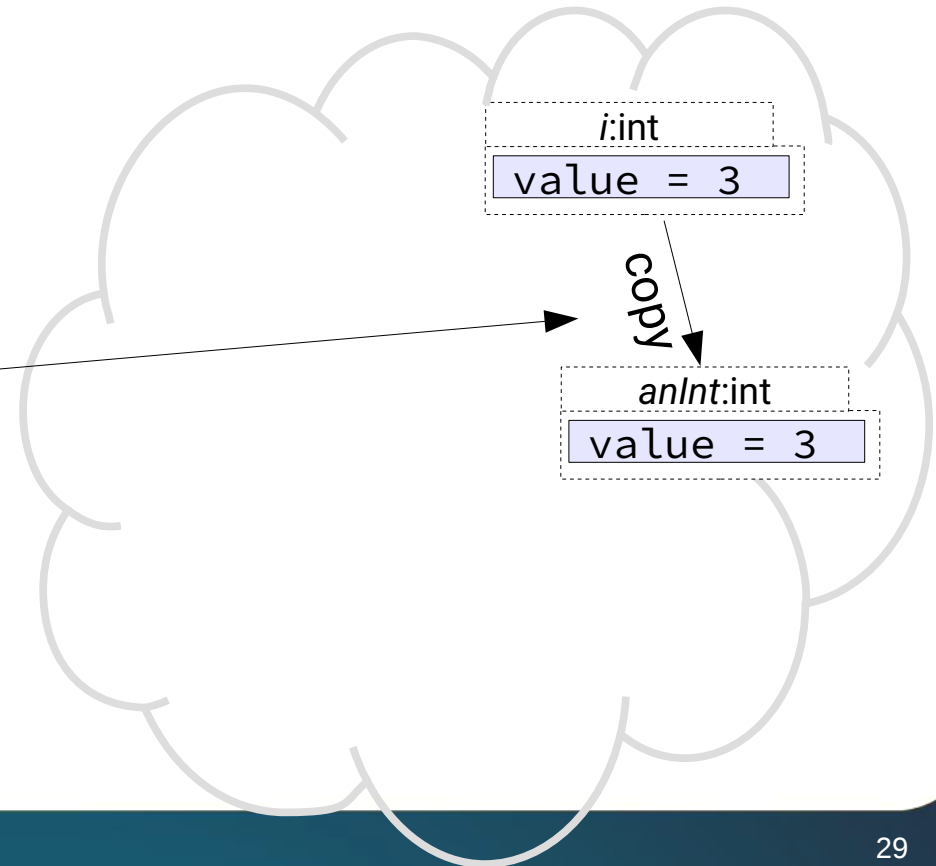
# Memory concerns...

## *passage par copie / valeur*

- Spécifié dans la déclaration & la définition d'une fonction

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```



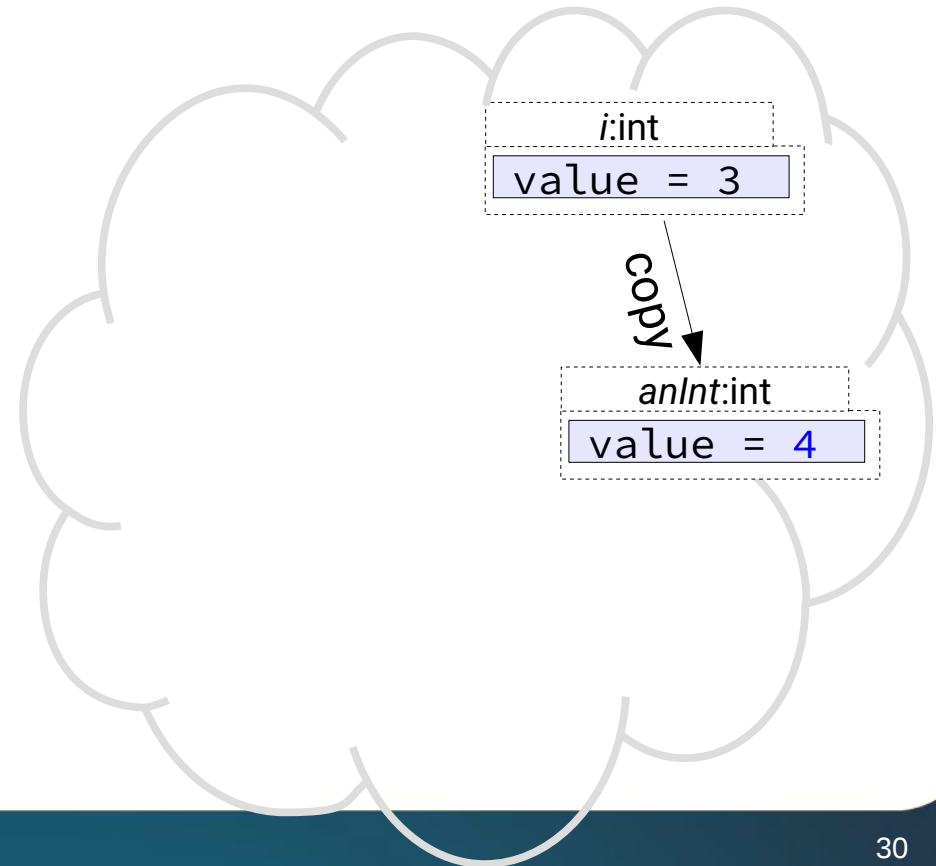
# Memory concerns...

## *passage par copie / valeur*

- Spécifié dans la déclaration & la définition d'une fonction

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```



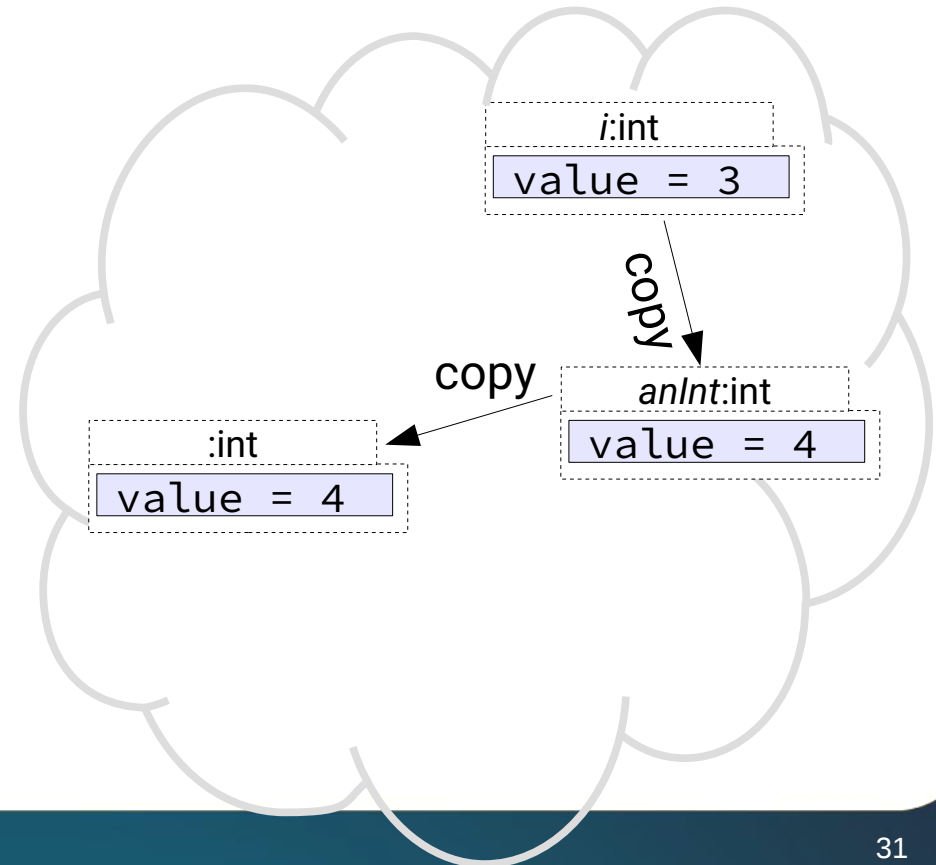
# Memory concerns...

## *passage par copie / valeur*

- Spécifié dans la déclaration & la définition d'une fonction

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```



# Memory concerns...

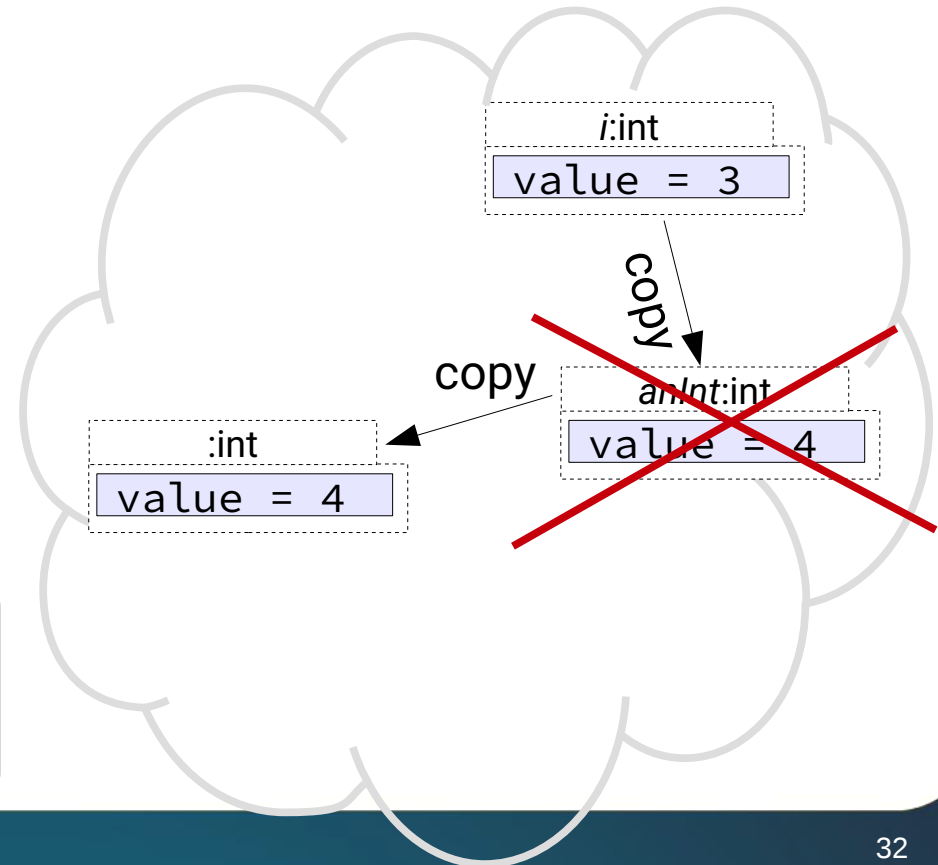
## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```

Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration





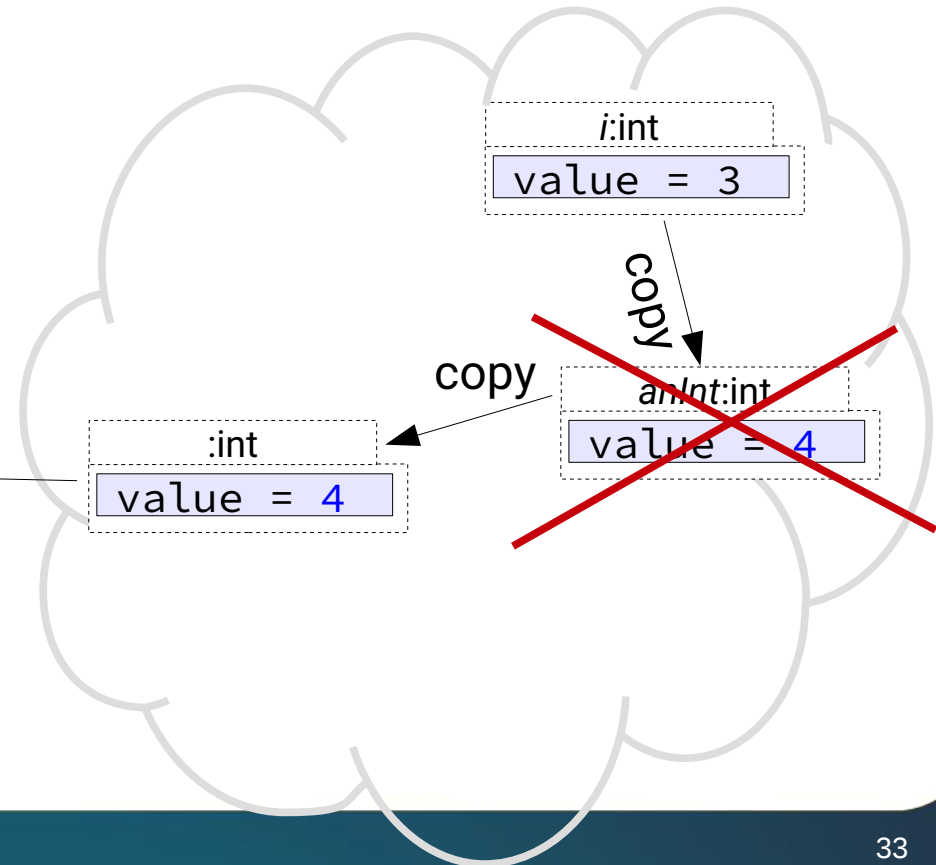
# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```



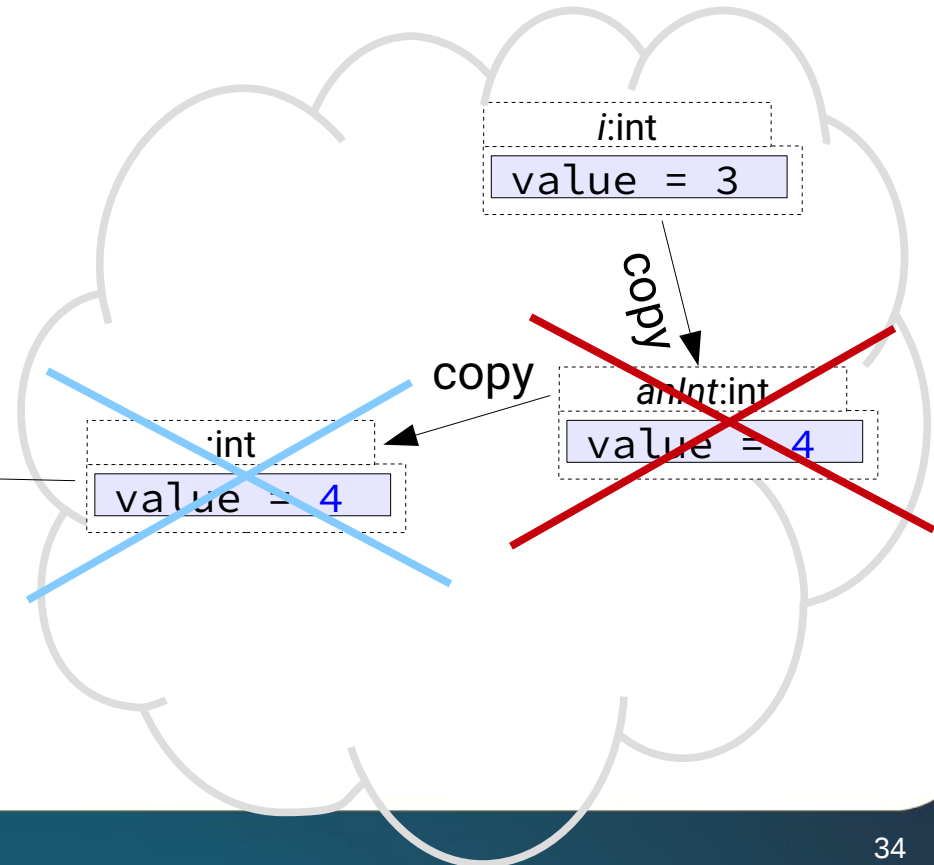
# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
main(){  
    int i;  
    i=3;  
    incremente(i);  
}
```



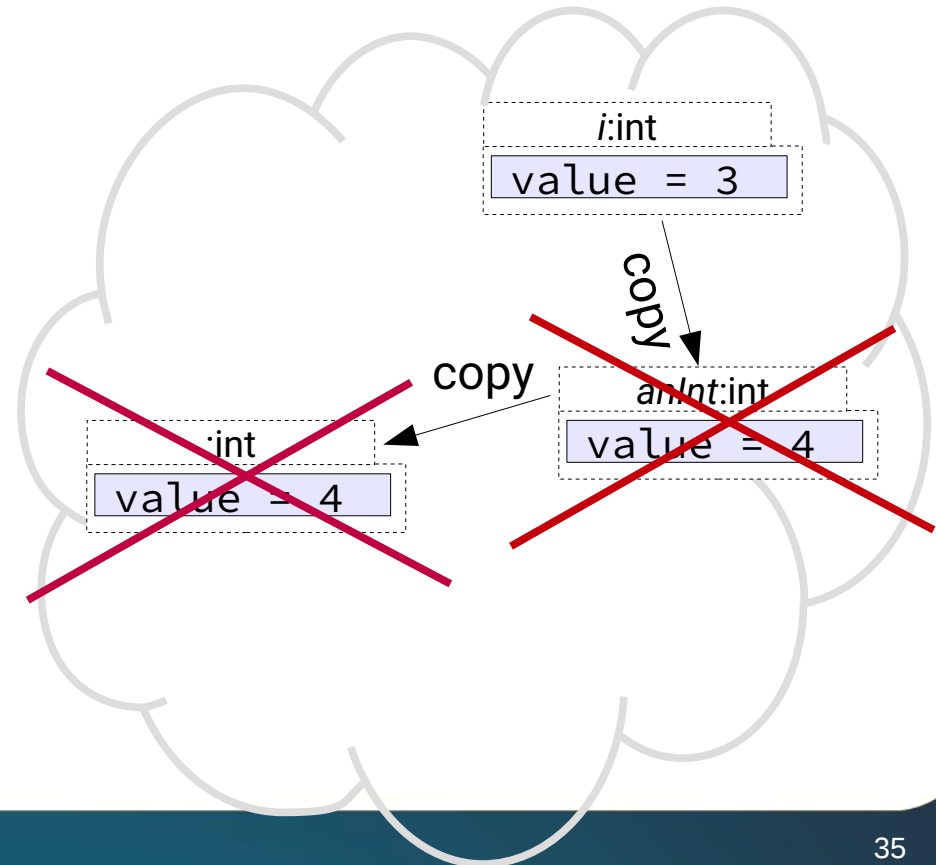
# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}  
  
main(){  
    int i;  
    i=3;  
    incremente(i);  
    std::cout<< i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
3  
jdeanton@FARCI:$
```



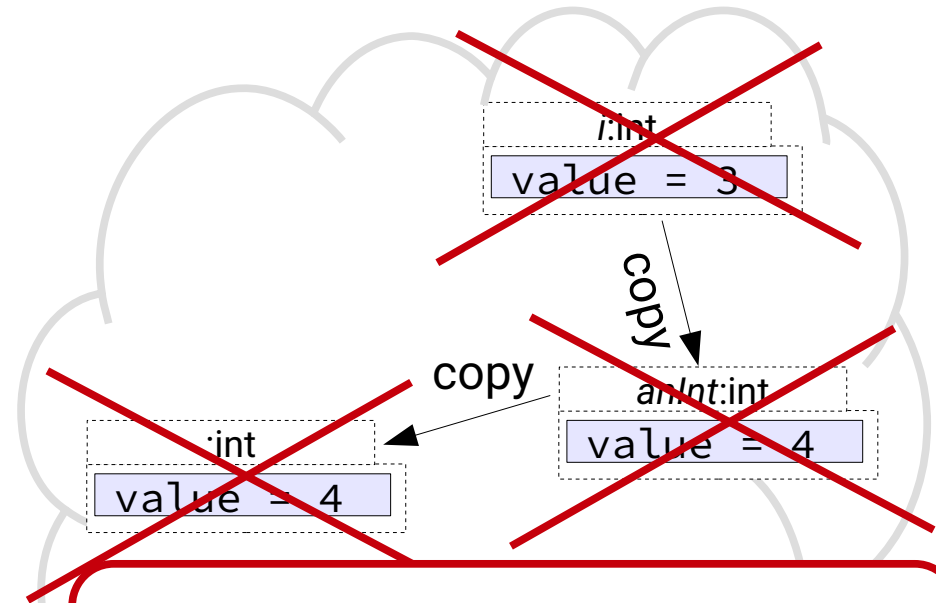
# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}  
  
main(){  
    int i;  
    i=3;  
    incremente(i);  
    std::cout<< i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
3  
jdeanton@FARCI:$
```



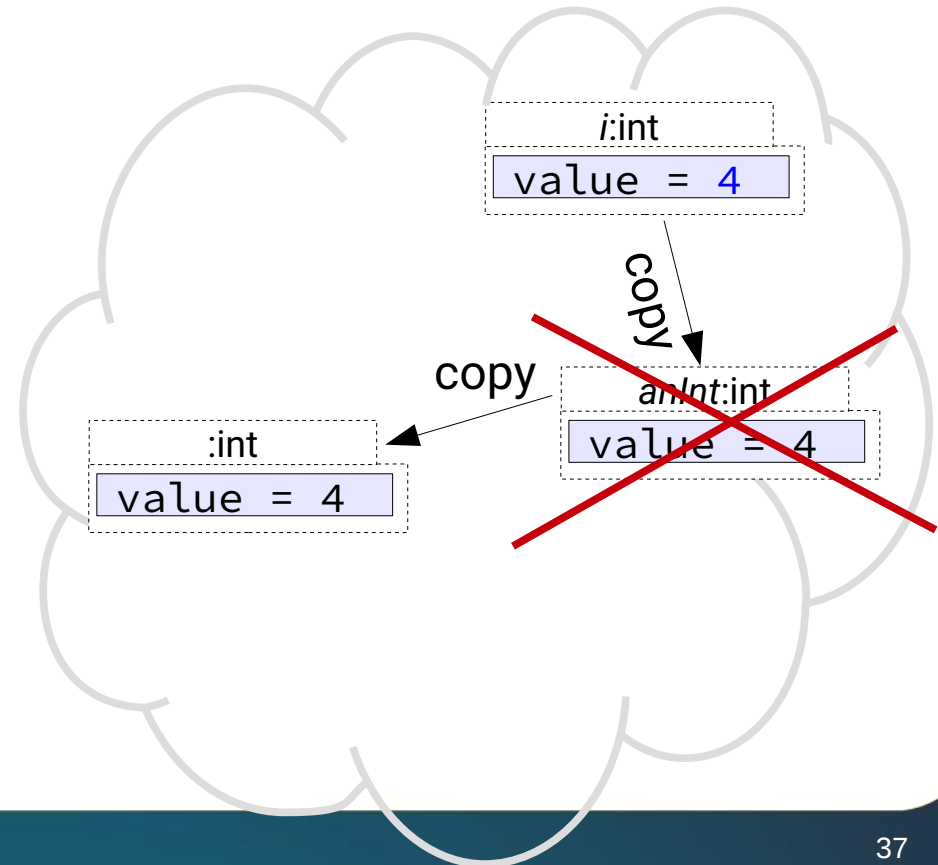
Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration

# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}  
  
main(){  
    int i;  
    i=3;  
    i = incremente(i);  
    std::cout<< i <<std::endl;  
}
```



# Memory concerns...

## *passage par copie / valeur*

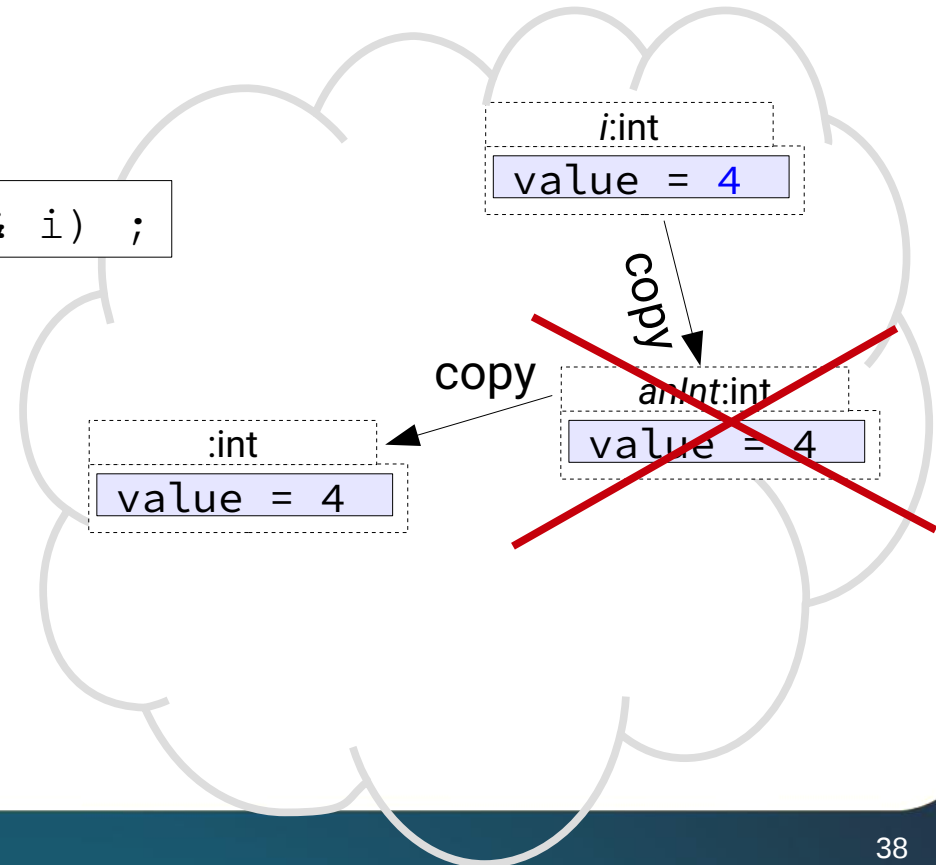
- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}
```

```
int operator=(const int& i) ;
```

*Operator d'affectation*

```
main(){  
    int i;  
    i=3;  
    i = incremente(i);  
    std::cout<< i <<std::endl;  
}
```

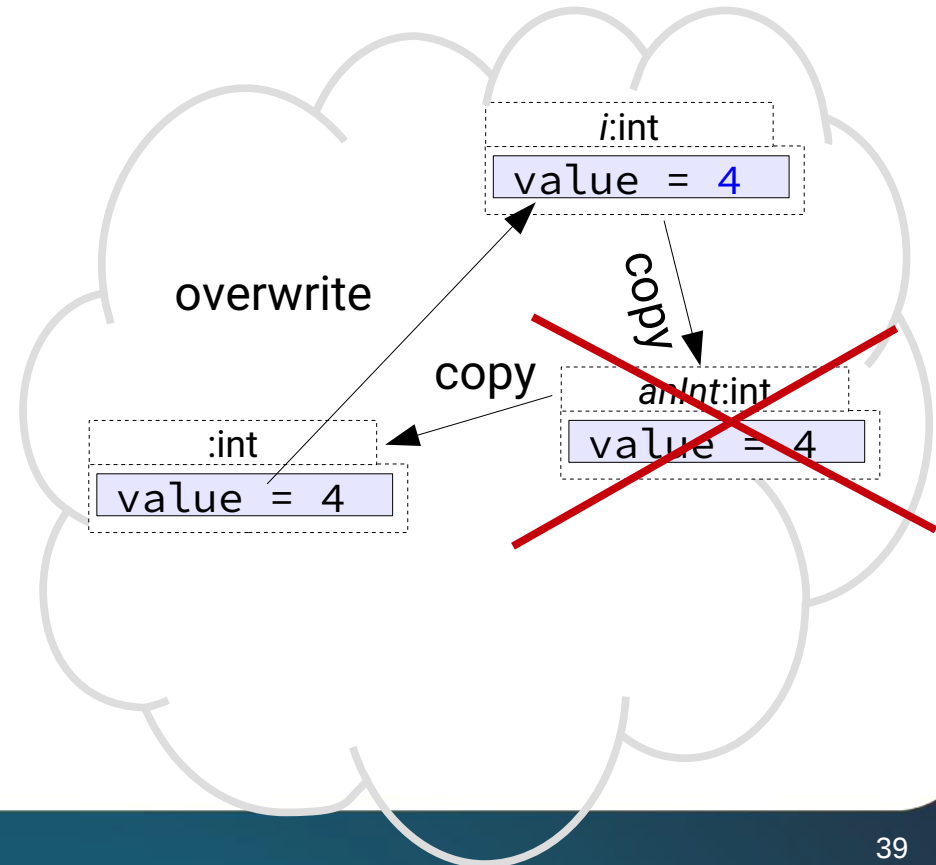


# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}  
  
main(){  
    int i;  
    i=3;  
    i = incremente(i);  
    std::cout<< i <<std::endl;  
}
```



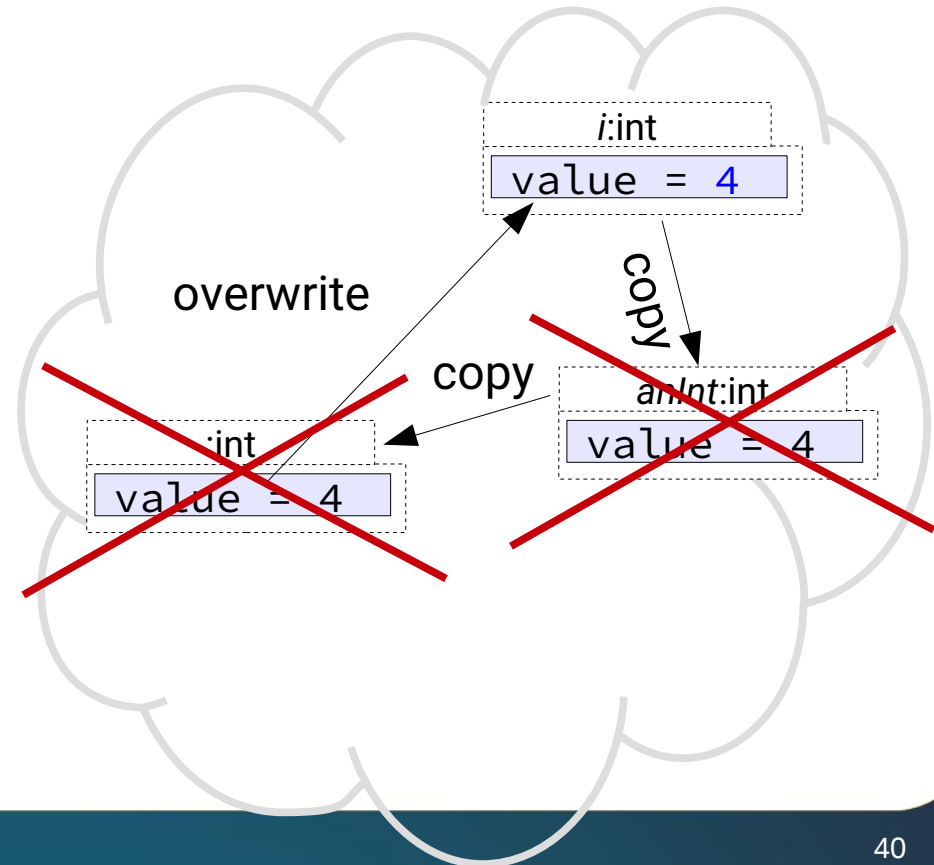
# Memory concerns...

## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){  
    anInt = anInt + 1;  
    return anInt;  
}  
  
main(){  
    int i;  
    i=3;  
    i = incremente(i);  
    std::cout<< i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
4  
jdeanton@FARCI:$
```





# Memory concerns...

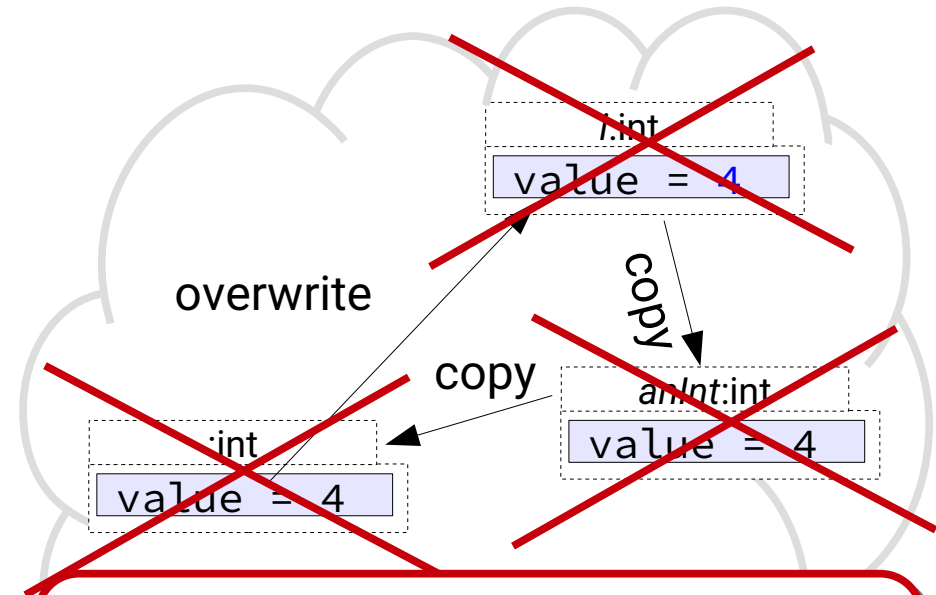
## *passage par copie / valeur*

- `anInt` est local à la fonction et détruit à la fin

```
int incremente(int anInt){
    anInt = anInt + 1;
    return anInt;
}

main(){
    int i;
    i=3;
    i = incremente(i);
    std::cout<< i <<std::endl;
}
```

```
jdeanton@FARCI:$/executable
4
jdeanton@FARCI:$
```



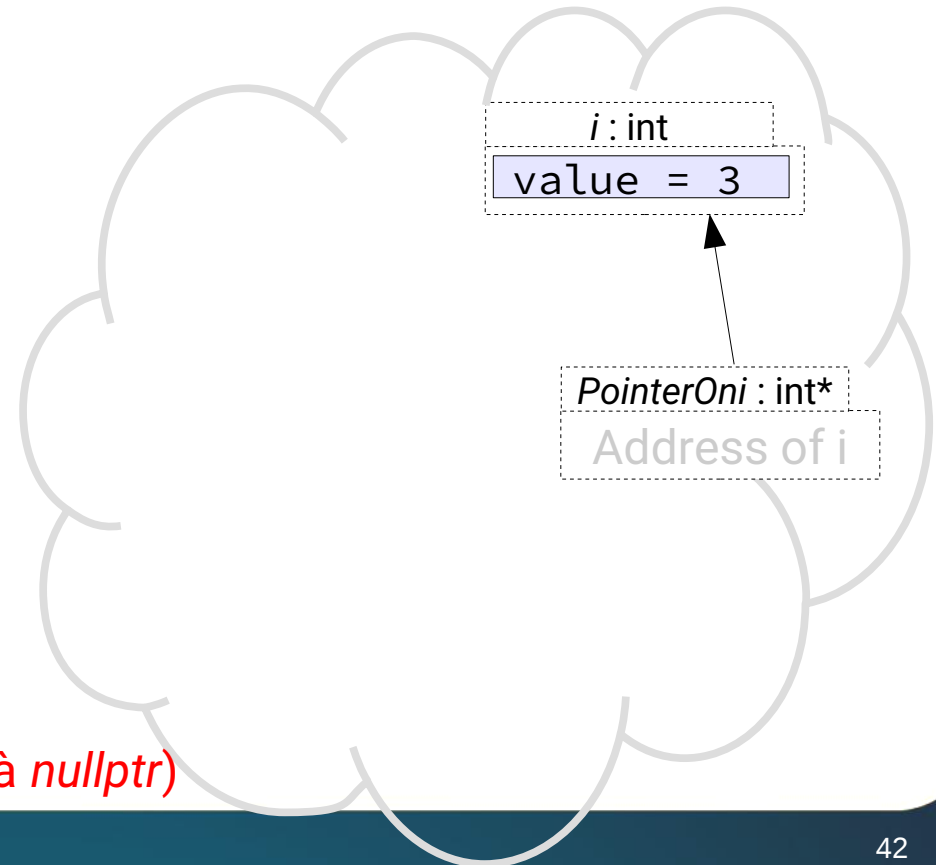
Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration

# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointer0ni = &i;  
}
```



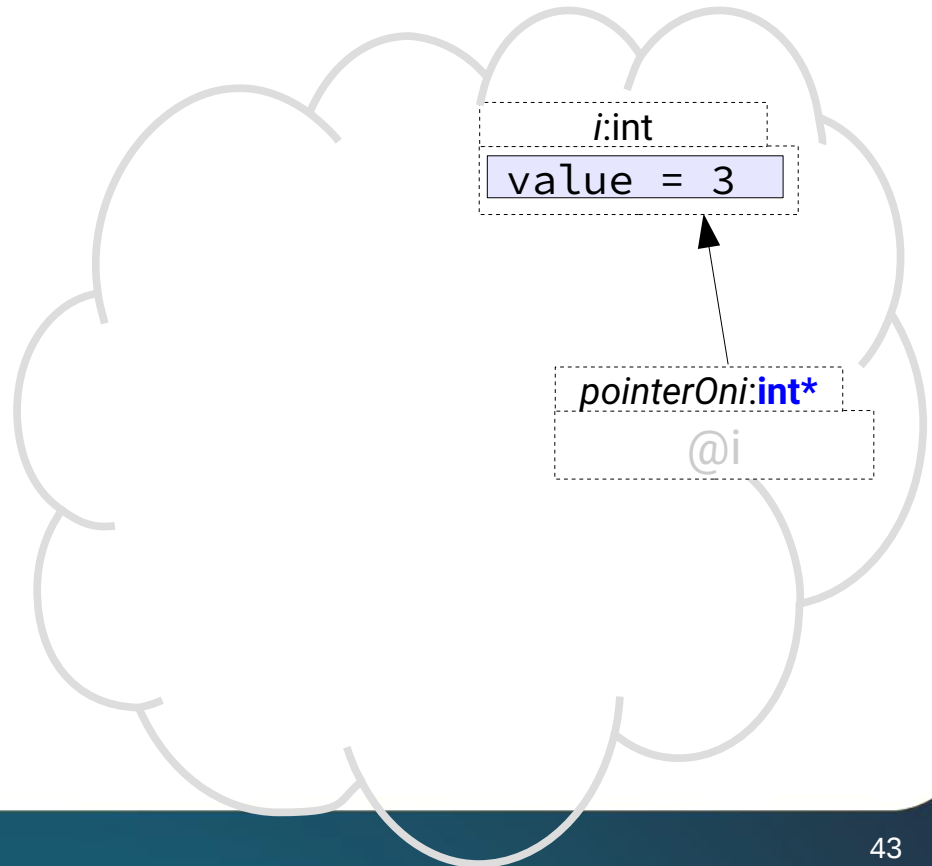
Un pointeur non initialisé est une erreur  
(contenue aléatoire, ne peut pas être testé à *nullptr*)

# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointerOni = &i;  
}
```



Type « pointeur sur entier »

# Memory concerns...

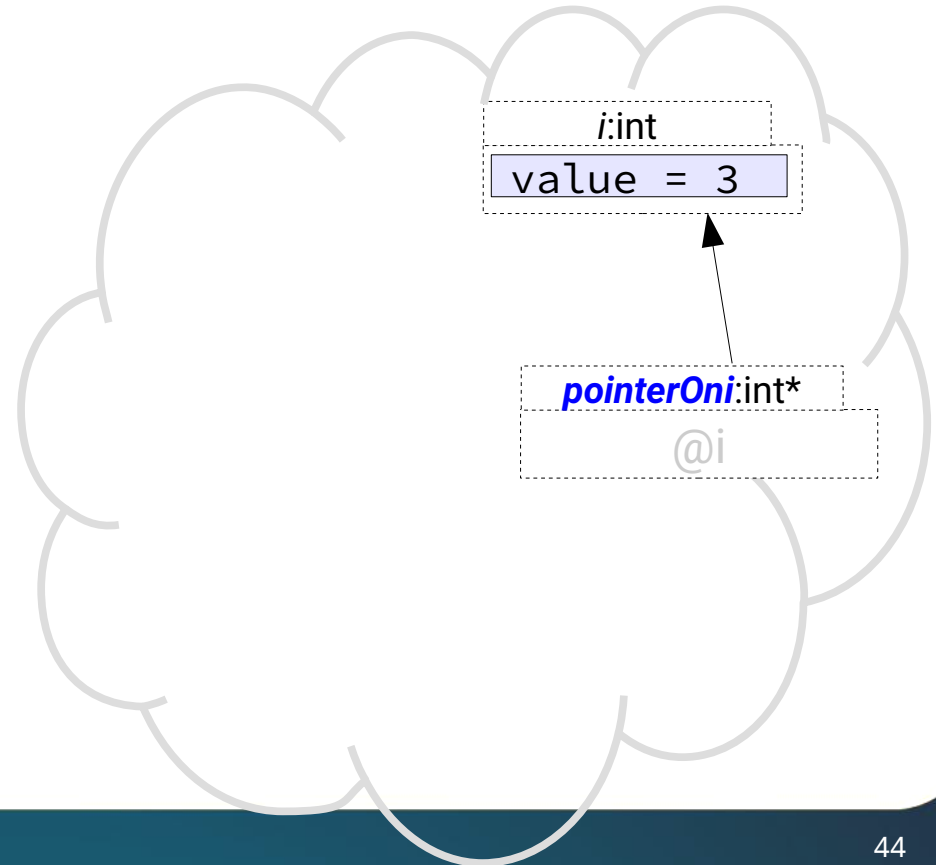
## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointer0ni = &i;  
}
```

Variable de type  
pointeur sur entier

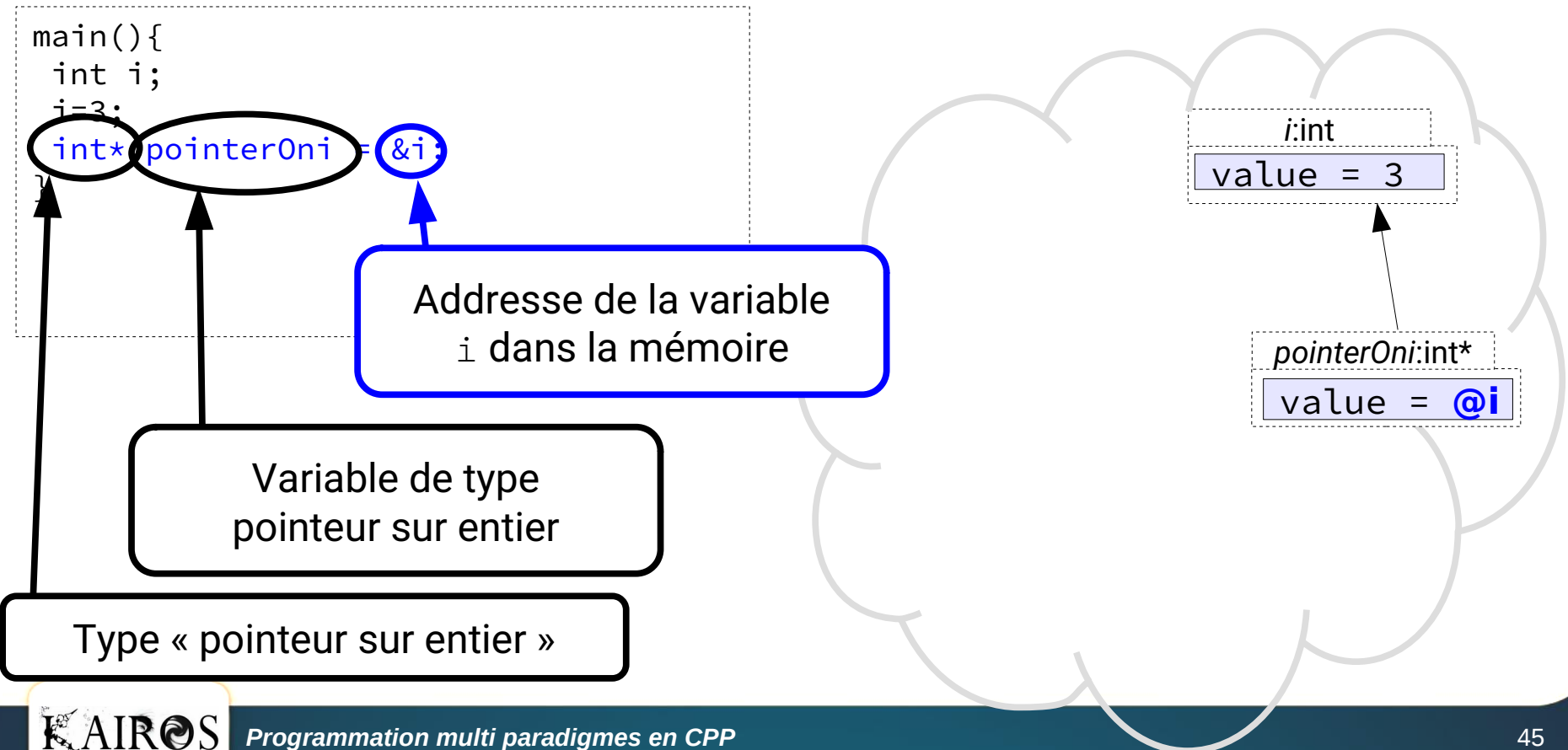
Type « pointeur sur entier »



# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

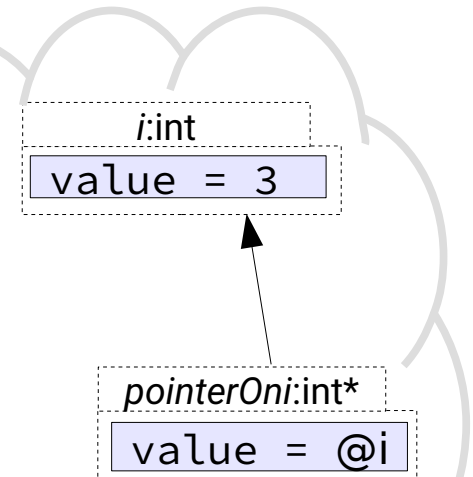


# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointerOni = &i;  
  std::cout << "address of i: " << pointerOni << std::endl;  
  std::cout << "address of i: " << &i << std::endl;  
}
```



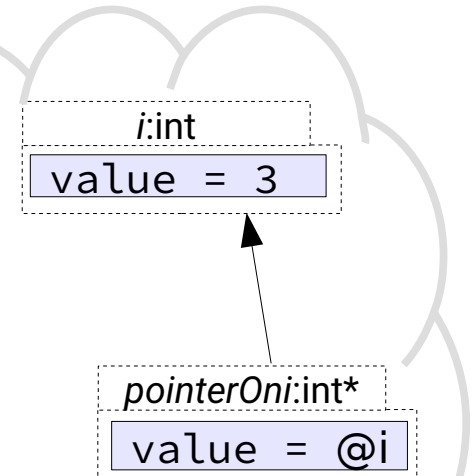
```
jdeanton@FARCI:$/executable  
address of i: 0x7fff0d584194  
address of i: 0x7fff0d584194  
jdeanton@FARCI:$
```

# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointerOni = &i;  
  std::cout << "address of i: " << pointerOni << std::endl;  
  std::cout << "address of i: " << &i << std::endl;  
}
```



La taille d'un pointeur  
Ne varie pas en fonction  
de la taille de l'objet pointé

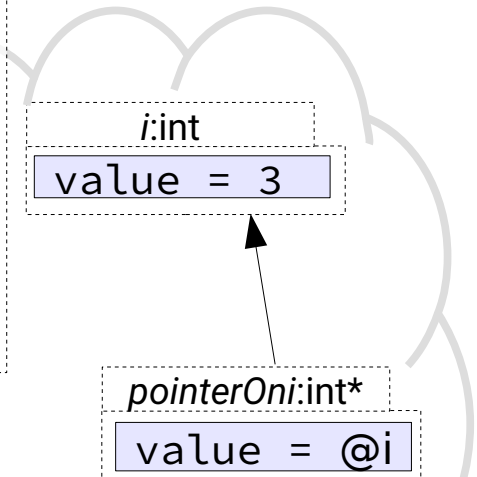
```
jdeanton@FARCI:$/executable  
address of i: 0x7fff0d584194  
address of i: 0x7fff0d584194  
jdeanton@FARCI:$
```

# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointer0ni = &i;  
  std::cout << "value of i: " << i << std::endl;  
  std::cout << "value of i: " << (*pointer0ni) << std::endl;  
}
```



```
jdeanton@FARCI:$/executable  
value of i: 3  
value of i: 3  
jdeanton@FARCI:$
```



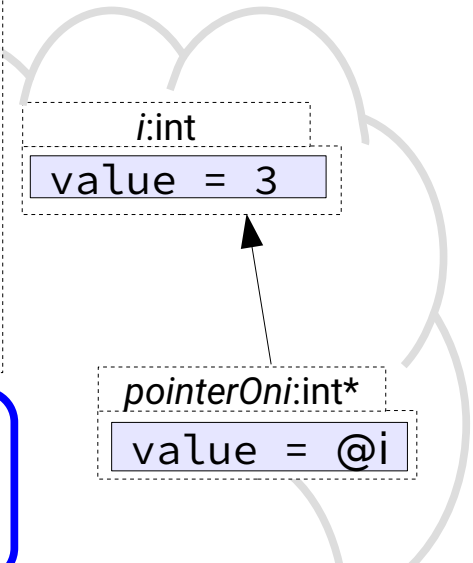
# Memory concerns...

## *notion de pointeur*

- Un pointeur est une variable dont le contenu spécifie un emplacement en mémoire

```
main(){  
  int i;  
  i=3;  
  int* pointerOni = &i;  
  std::cout << "value of i: " << i << std::endl;  
  std::cout << "value of i: " << (*pointerOni) << std::endl;  
}
```

Accès à la variable / objet pointé  
(déréférencement de pointeur)



```
jdeanton@FARCI:$/executable  
value of i: 3  
value of i: 3  
jdeanton@FARCI:$
```



**Verbe** [modifier le wikicode]

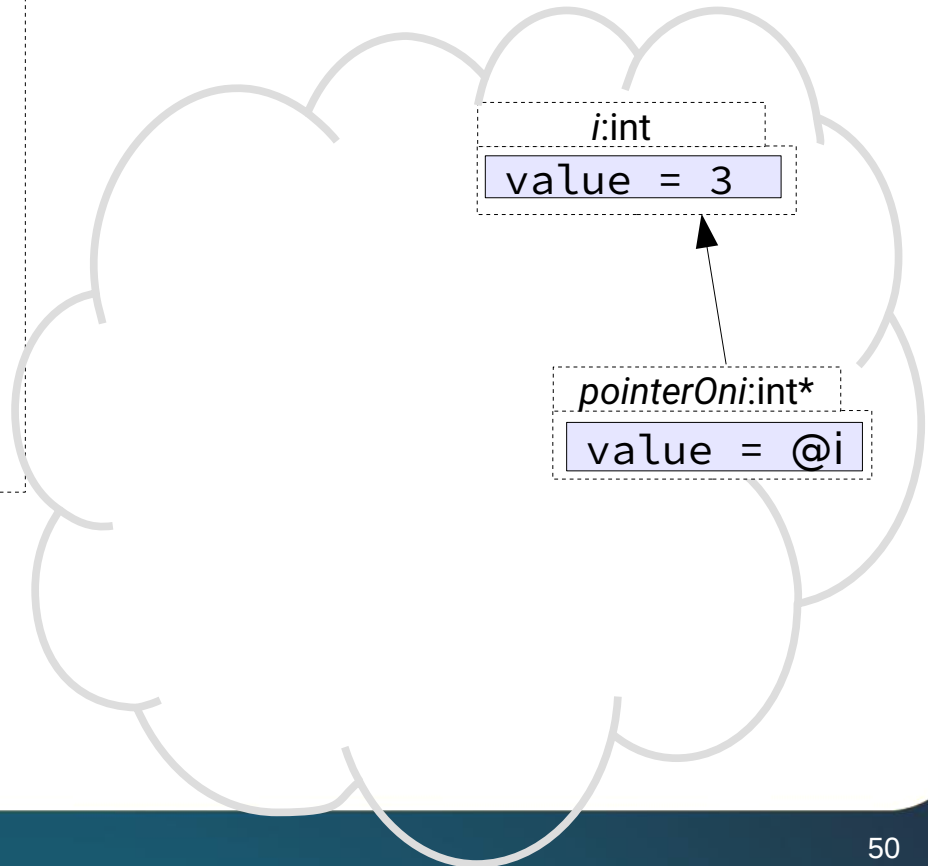
**déréférencer** \de.ʁe.fe.ʁã.se\ transitif 1<sup>er</sup> groupe (conjugaison)

- Ne plus **référencer**.
- (Programmation informatique) Obtenir l'objet pointé par un **pointeur**.

# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
}
```

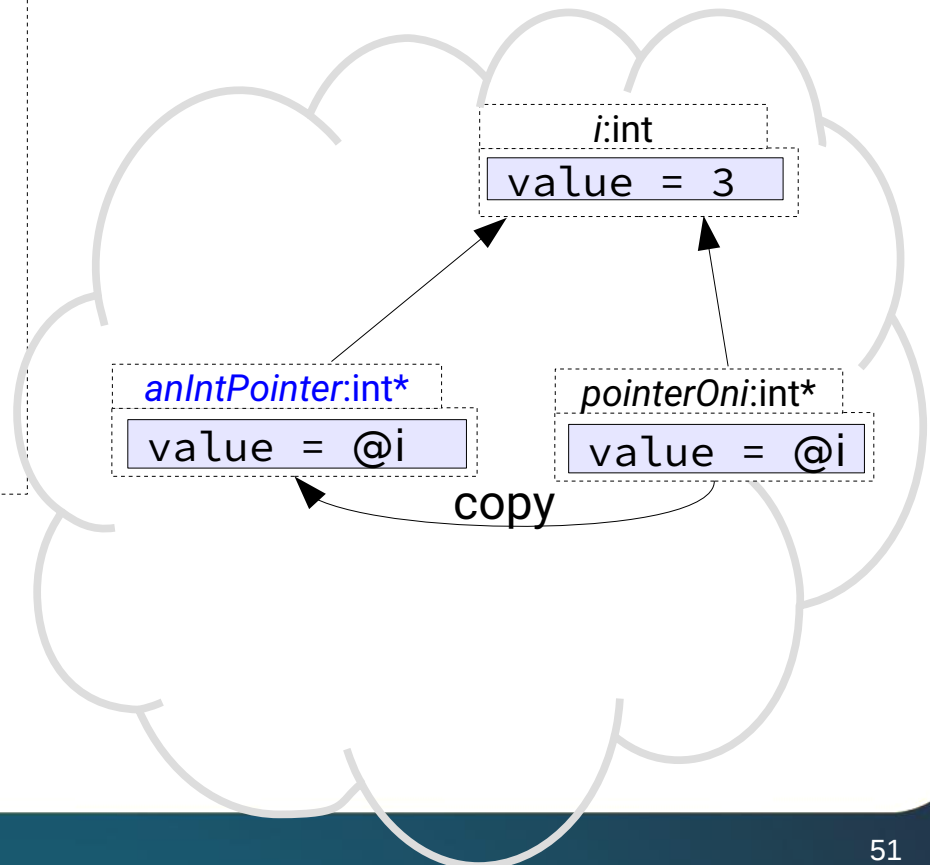


# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}
```

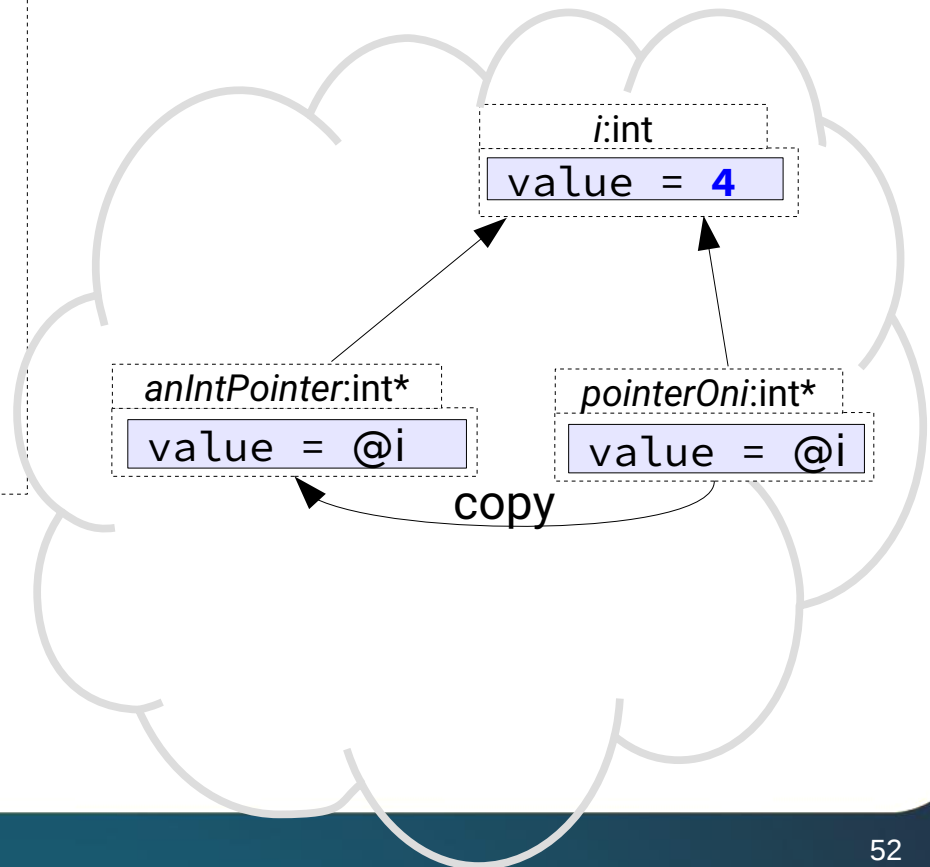
```
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
}
```



# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

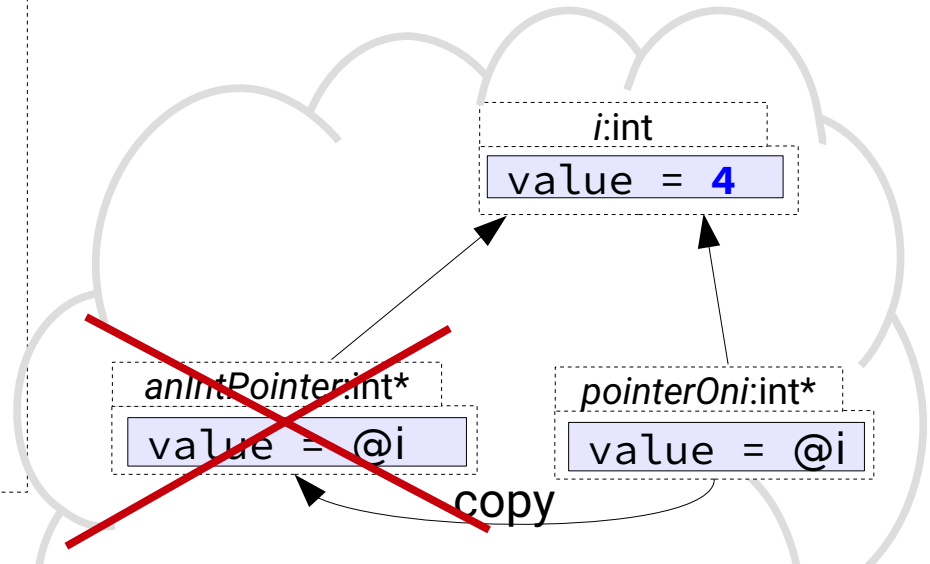
```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
}
```



# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
}
```



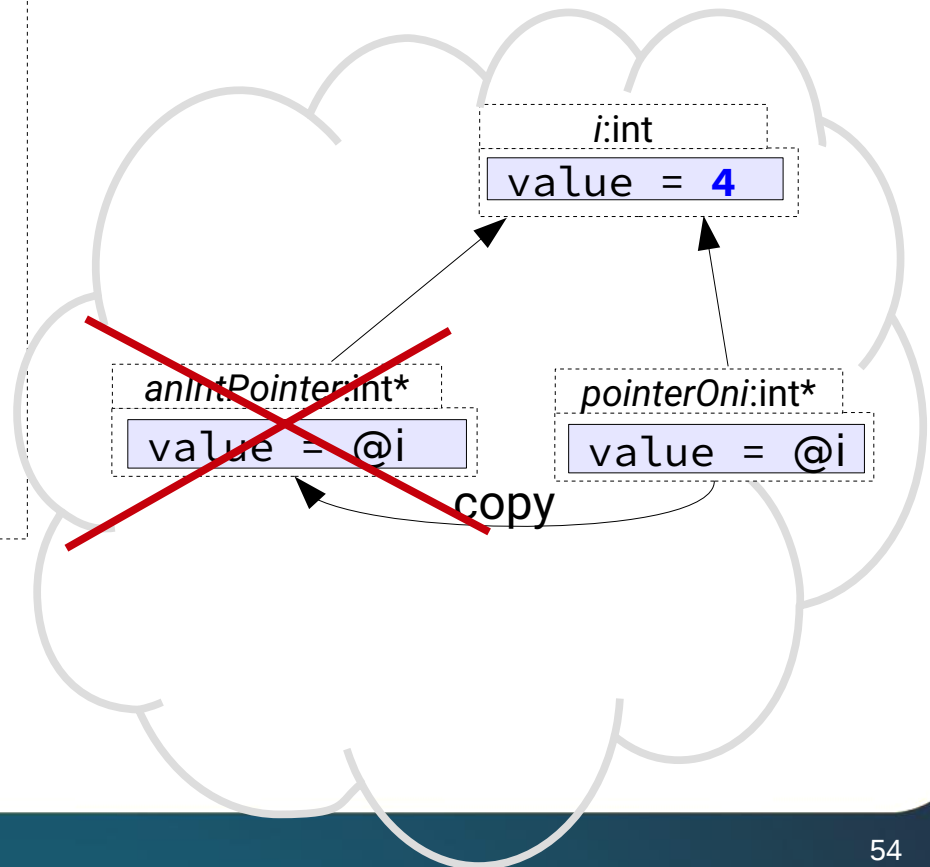
Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration

# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
    std::cout << i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
4  
jdeanton@FARCI:$
```

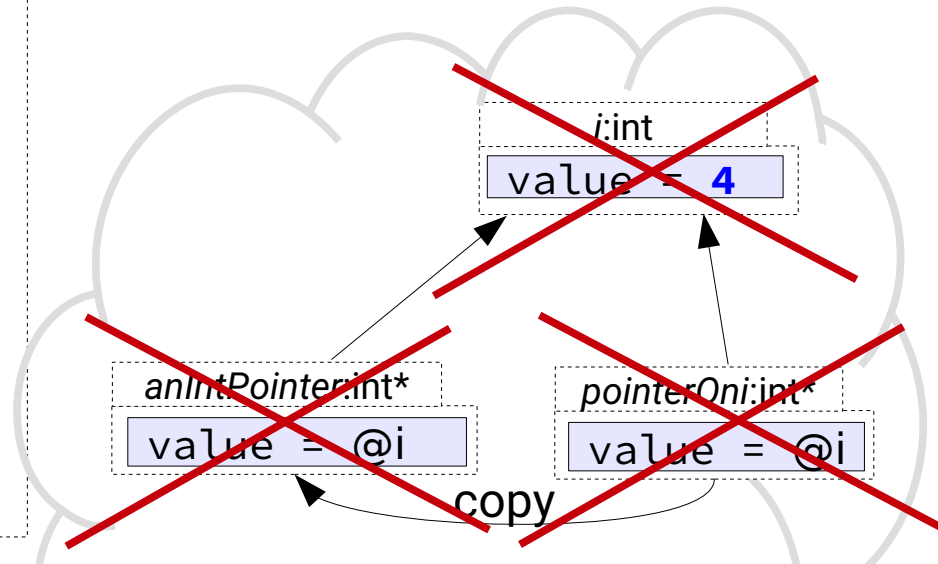


# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr) + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
    std::cout << i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
4  
jdeanton@FARCI:$
```

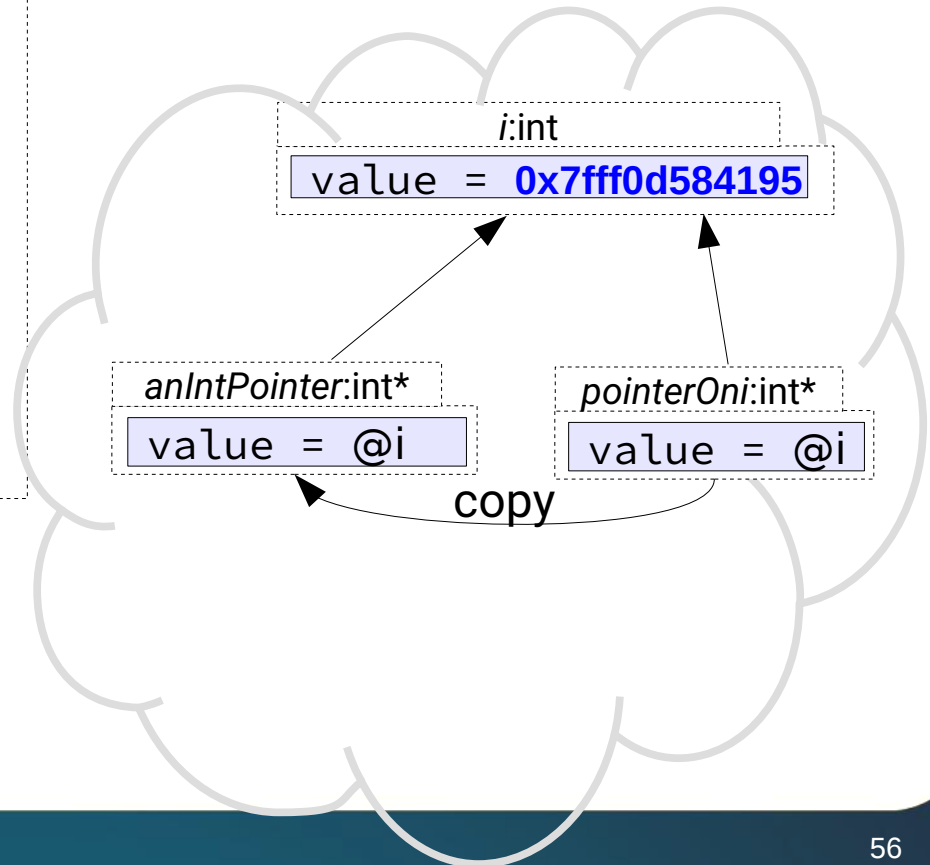


Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration

# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = anIntPtr + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointerOni = &i;  
    incremente(pointerOni);  
}
```





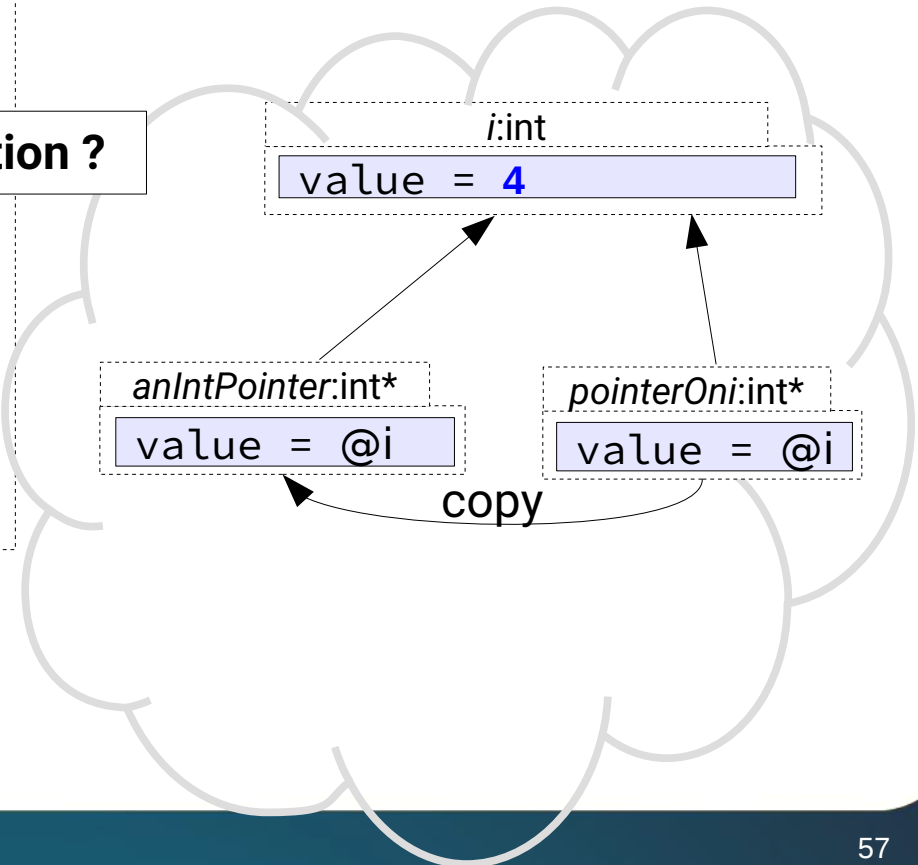
# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    (*anIntPtr) = (*anIntPtr)+1;  
    return;  
}
```

**Quel problème dans cette fonction ?**

```
main(){  
    int i;  
    i=3;  
    int* pointerOni = &i;  
    incremente(pointerOni);  
}
```



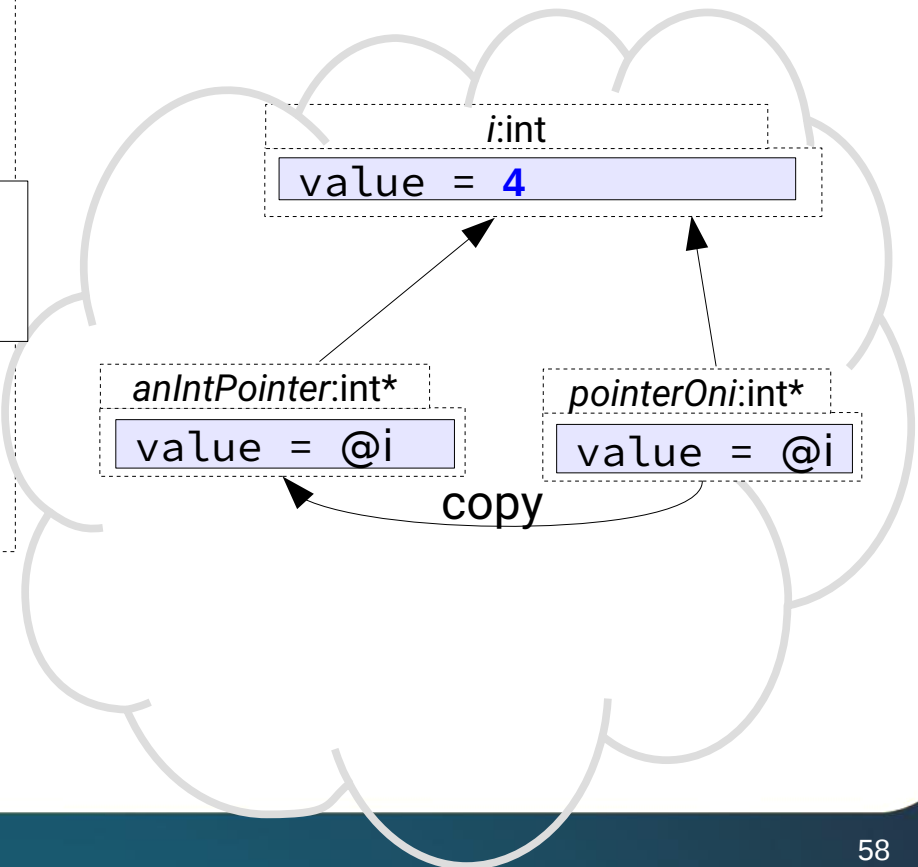
# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    if (anIntPtr == nullptr){exit(-1)}  
    (*anIntPtr) = (*anIntPtr)+1;  
    return;  
}
```

**Toujours tester un pointer à nullptr avant de le déréférencer**

```
main(){  
    int i;  
    i=3;  
    int* pointerOni = &i;  
    incremente(pointerOni);  
}
```



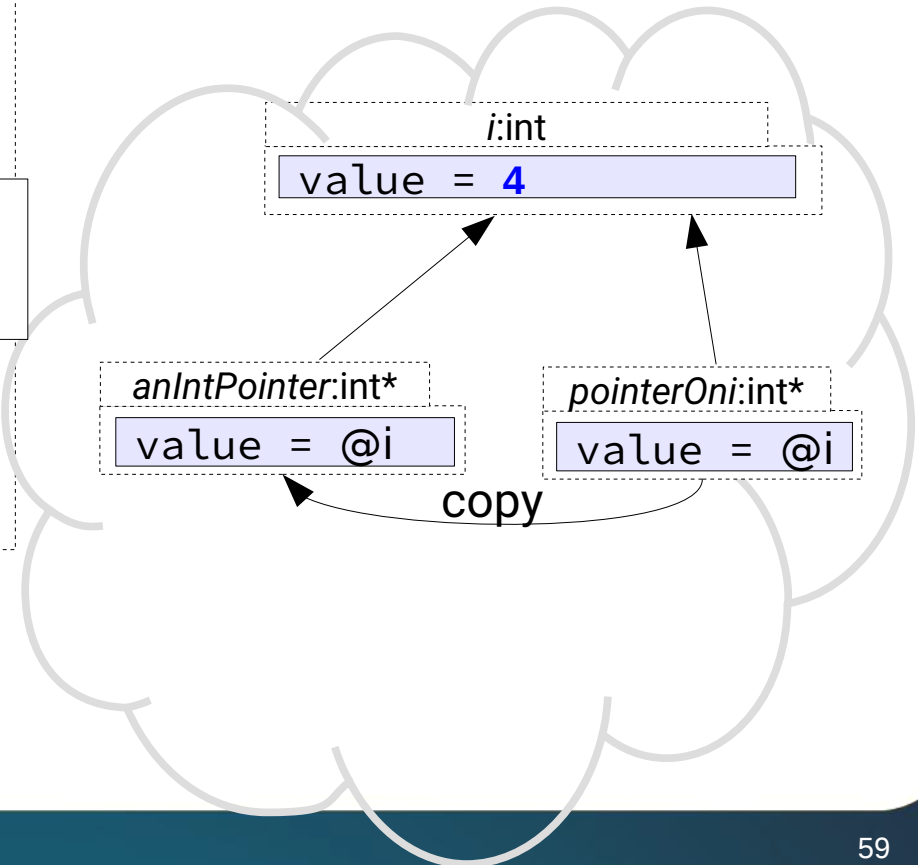
# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction

```
void incremente(int* anIntPtr){  
    if (anIntPtr == nullptr){exit(-1)}  
    (*anIntPtr) = (*anIntPtr)+1;  
    return;  
}
```

**Toujours** tester un pointer à nullptr avant de le déréférencer

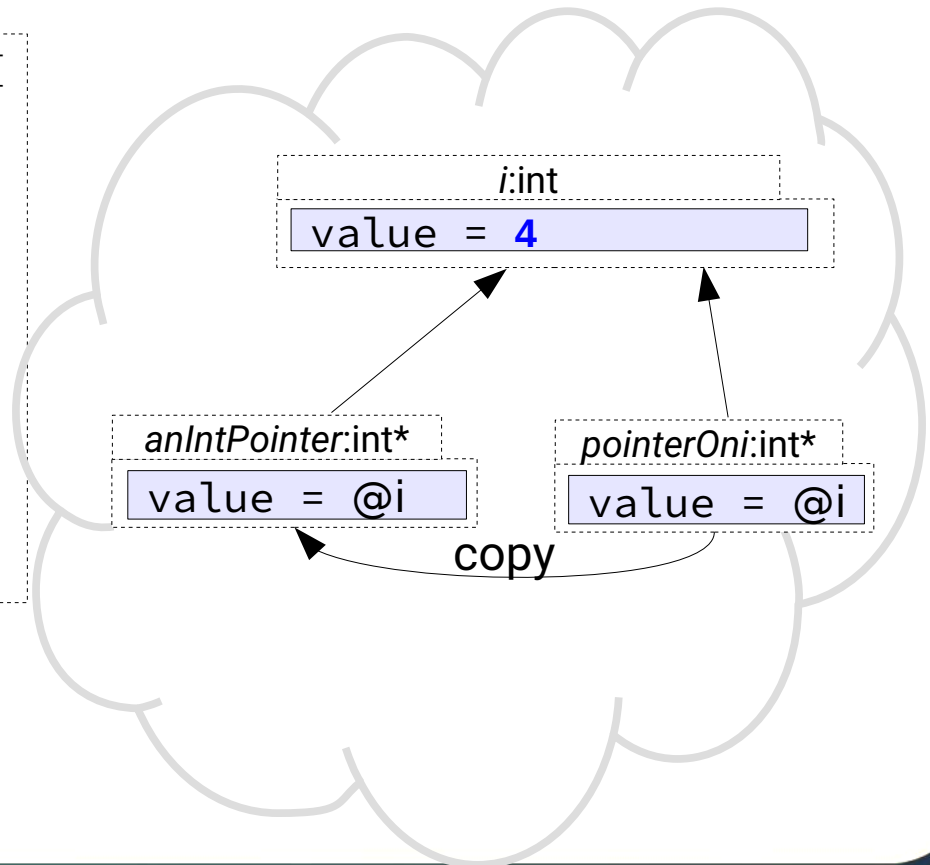
```
main(){  
    int i;  
    i=3;  
    int* pointerOni = &i;  
    incremente(pointerOni);  
}
```



# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction
- Un pointeur constant empêche la modification de l'objet pointé

```
void incremente(const int* anIntPtr){  
    if (anIntPtr == nullptr){exit(-1)}  
    (*anIntPtr) = (*anIntPtr)+1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int* pointerOni = &i;  
    incremente(pointerOni);  
}
```



# Memory concerns...

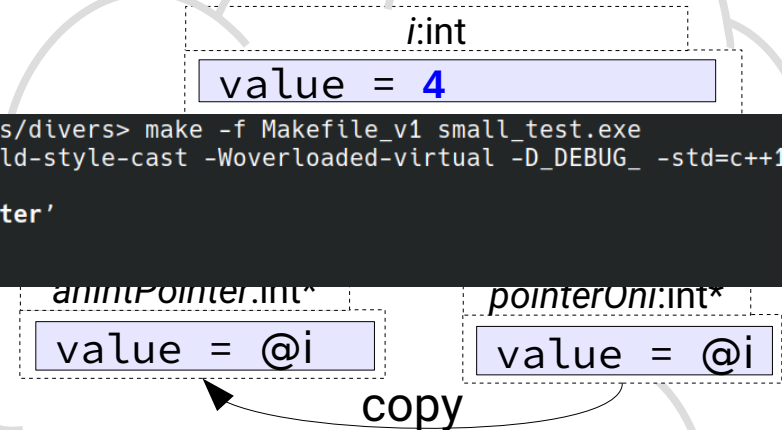
- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction
- Un pointeur constant empêche la modification de l'objet pointé

```
void incremente(const int* anIntPtr){
    if (anIntPtr == nullptr){exit(-1)}
    (*anIntPtr) = (*anIntPtr)+1;
    return;
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
small_test.cpp: In function 'void incremente(const int*)':
small_test.cpp:16:18: error: assignment of read-only location '* aPointer'
```

```
16 |     (*aPointer) = (*aPointer) + 1;
    |     ~~~~~^~~~~~
```

```
int* pointerOni = &i;
incremente(pointerOni);
}
```



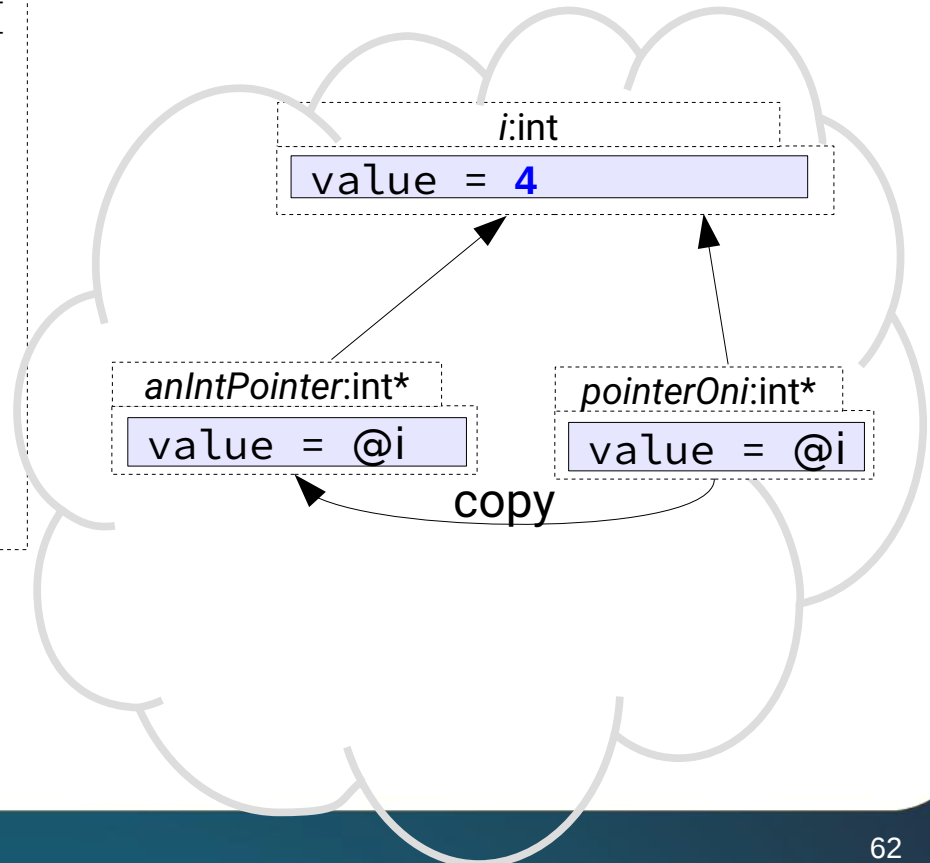
# Memory concerns...

- Le passage par pointeur est spécifié dans la définition et la déclaration de la fonction
- Un pointeur constant empêche la modification de l'objet pointé, **pas du pointeur lui même**

```
void incremente(const int* anIntPtr){  
    if (anIntPtr == nullptr){exit(-1)}  
    anIntPtr = anIntPtr+1;  
    return;  
}
```



```
main(){  
    int i;  
    i=3;  
    int* pointer0ni = &i;  
    incremente(pointer0ni);  
}
```



# Memory concerns...

## *notion de référence*

- Une référence est un alias (un autre nom) d'une variable
- Une référence ne peut être affectée qu'à sa création

```
main(){  
  int i;  
  i=3;  
}
```

i:int

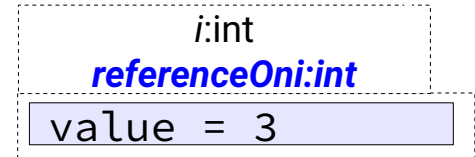
value = 3

# Memory concerns...

## *notion de référence*

- Une référence est un alias (un autre nom) d'une variable
- Une référence ne peut être affectée qu'à sa création

```
main(){  
  int i;  
  i=3;  
  int& referenceOni = i;  
}
```

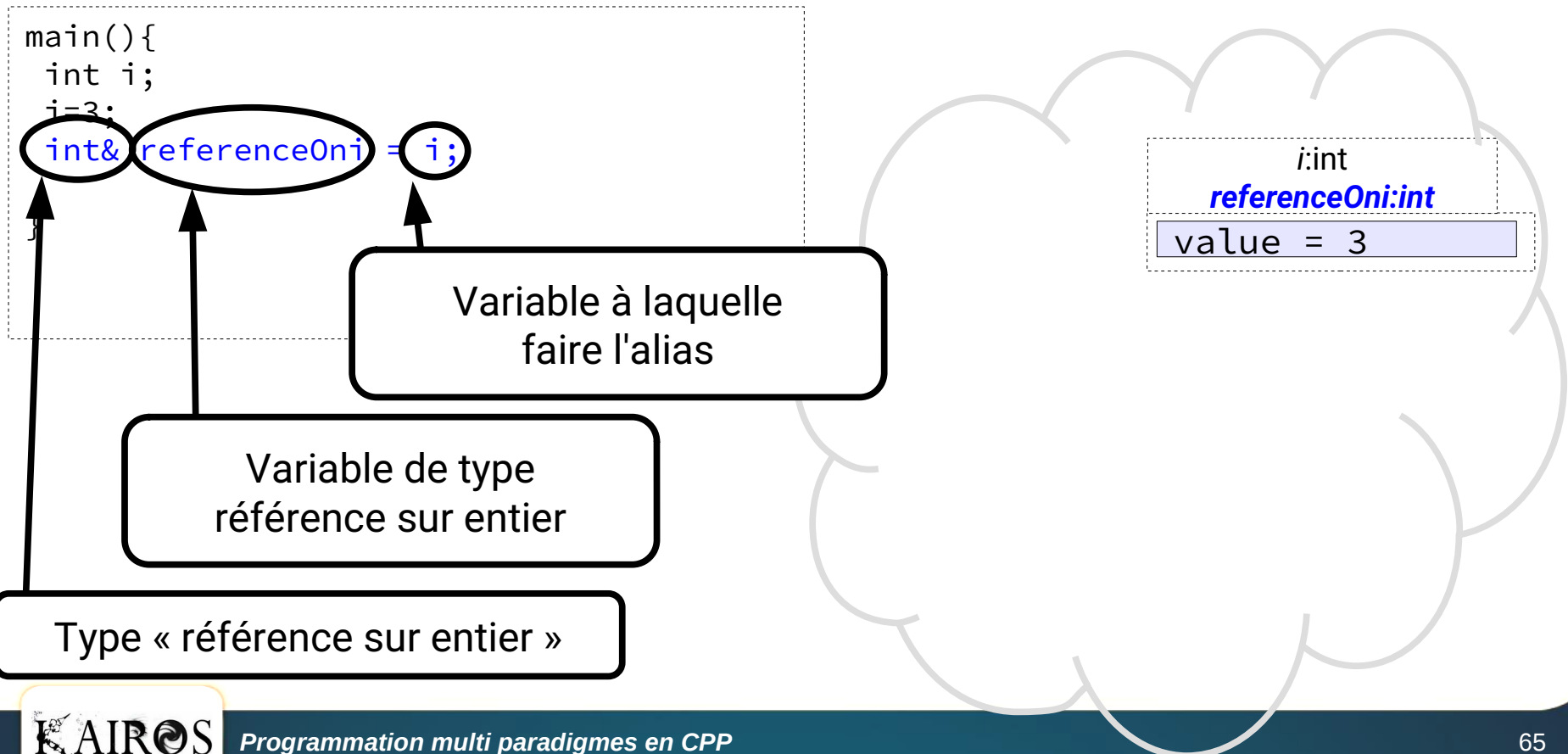




# Memory concerns...

## *notion de référence*

- Une référence est un alias (un autre nom) d'une variable
- Une référence ne peut être affectée qu'à sa création



# Memory concerns...

- Une référence est un alias (un autre nom) d'une variable
- Une référence ne peut être affectée qu'à sa création

```
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    std::cout << "address of i: " << &reference0ni << std::endl;  
    std::cout << "address of i: " << &i << std::endl;  
}
```

i:int  
reference0ni:int  
value = 3

```
jdeanton@FARCI:$/executable  
address of i: 0x7fff0d5ef587  
address of i: 0x7fff0d5ef587  
jdeanton@FARCI:$
```

Utiliser le nom ou la référence  
revient strictement au même

# Memory concerns...

- Une référence est un alias (un autre nom) d'une variable
- Une référence ne peut être affectée qu'à sa création

```
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    std::cout << "value of i: " << reference0ni << std::endl;  
    std::cout << "value of i: " << i << std::endl;  
}
```

i:int  
reference0ni:int  
value = 3

```
jdeanton@FARCI:$/executable  
value of i: 3  
value of i: 3  
jdeanton@FARCI:$
```

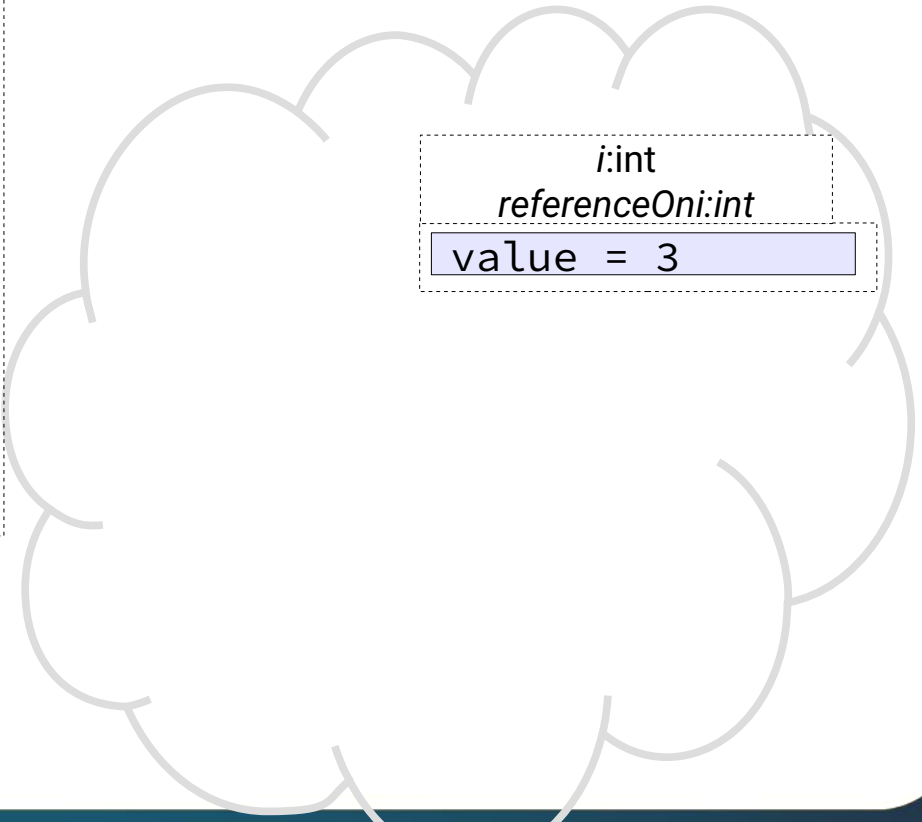
Utiliser le nom ou la référence  
revient strictement au même

# Memory concerns...

## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    incremente(i);  
}
```



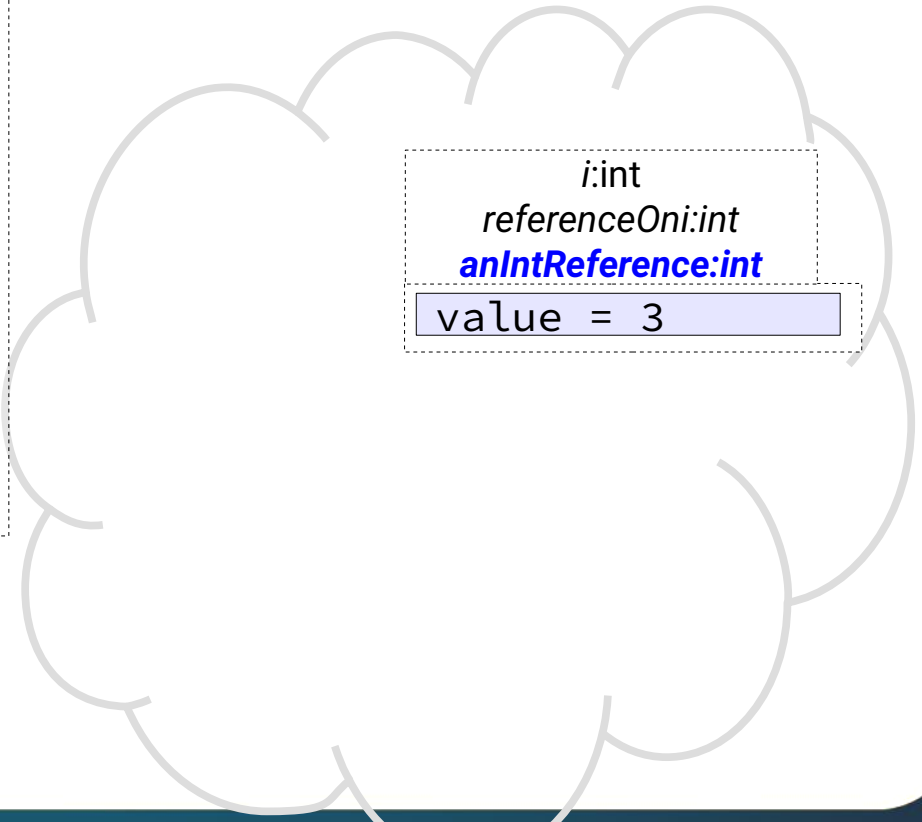
i:int  
reference0ni:int  
value = 3

# Memory concerns...

## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    incremente(i);  
}
```



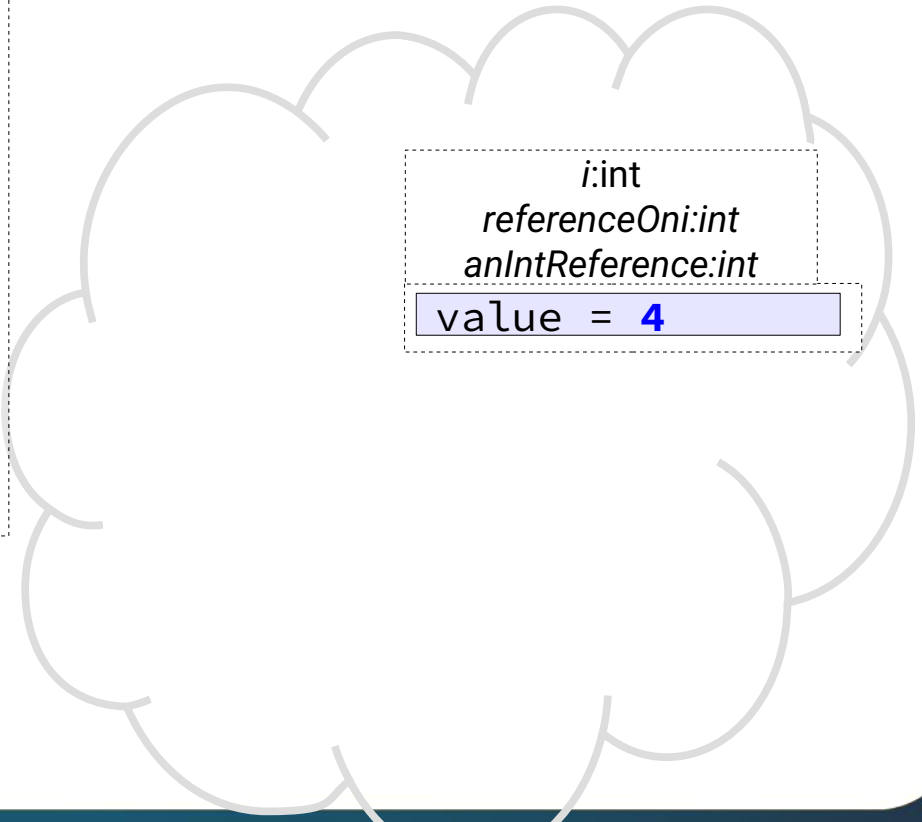
i:int  
reference0ni:int  
**anIntReference:int**  
value = 3

# Memory concerns...

## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}  
  
main(){  
    int i;  
    i=3;  
    int& referenceOni = i;  
    incremente(i);  
}
```



i:int  
referenceOni:int  
anIntReference:int  
value = 4

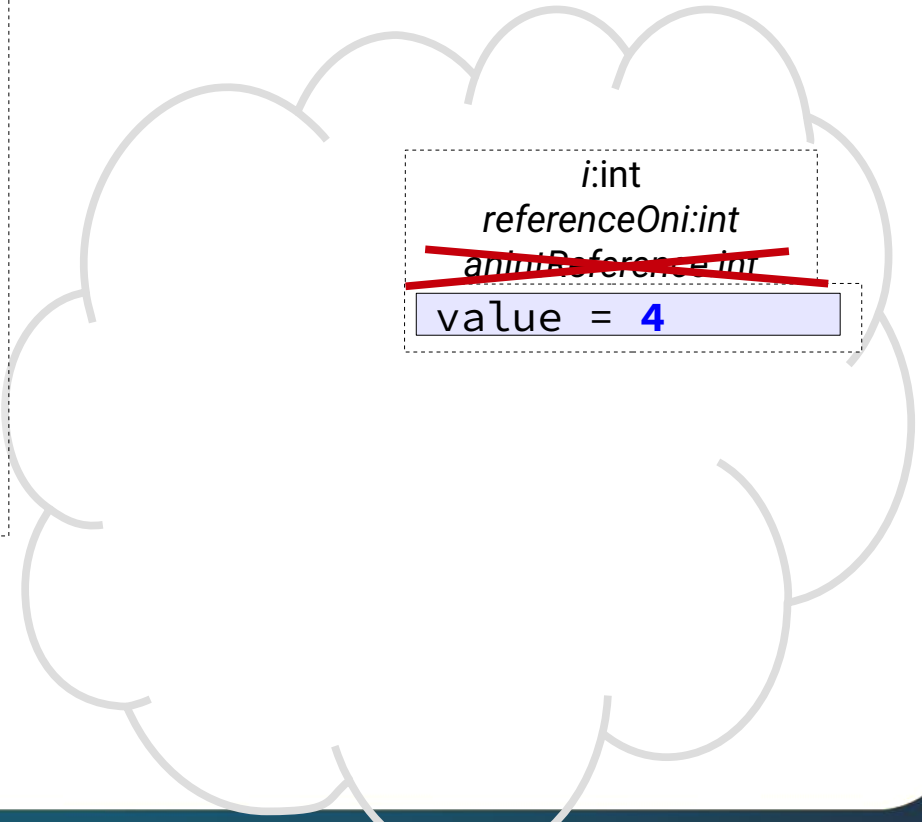
# Memory concerns...

## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}
```

```
main(){  
    int i;  
    i=3;  
    int& referenceOni = i;  
    incremente(i);  
}
```



i:int  
referenceOni:int  
~~anIntReference:int~~  
value = 4

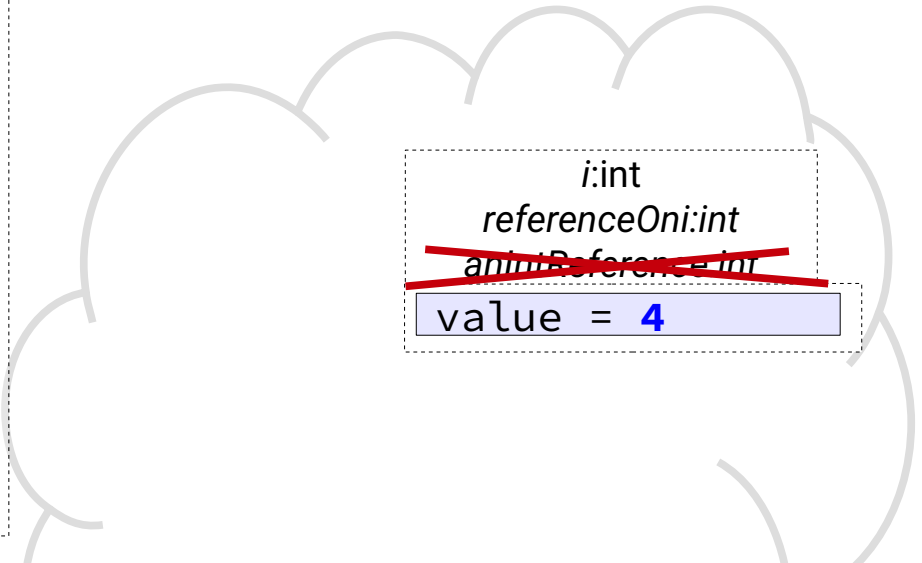
# Memory concerns...

## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}
```

```
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    incremente(i);  
}
```



i:int  
reference0ni:int  
~~anIntReference: int~~  
value = 4

Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration



# Memory concerns...

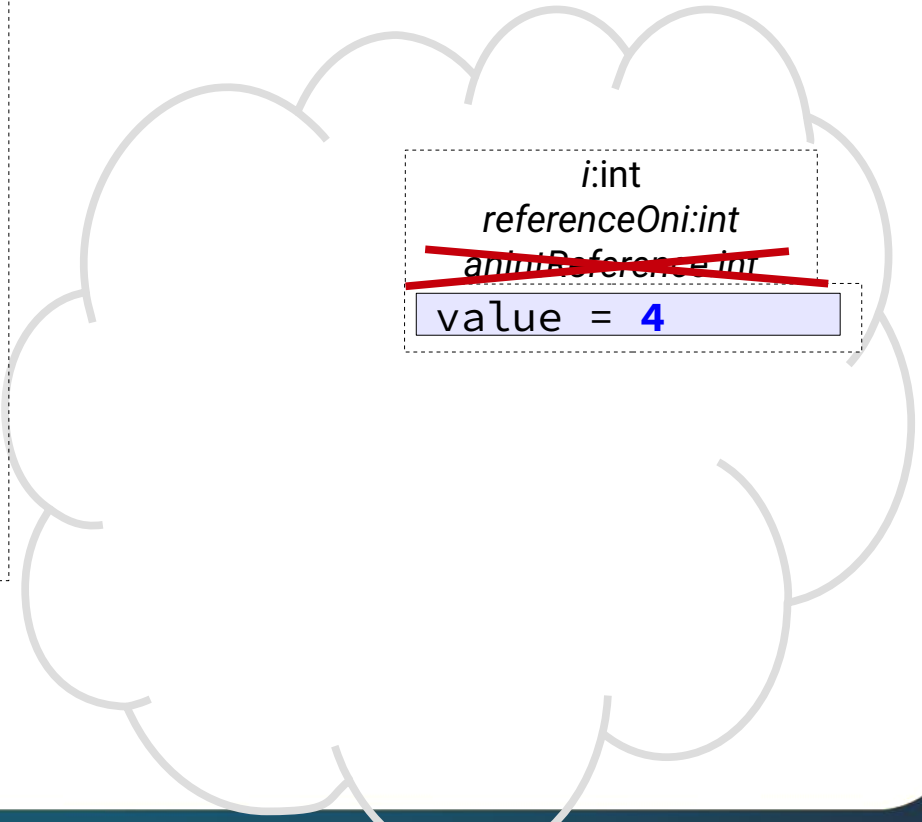
## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}
```

```
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    incremente(i);  
    std::cout << i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
4  
jdeanton@FARCI:$
```



i:int  
reference0ni:int  
~~anIntReference:int~~  
value = 4

# Memory concerns...

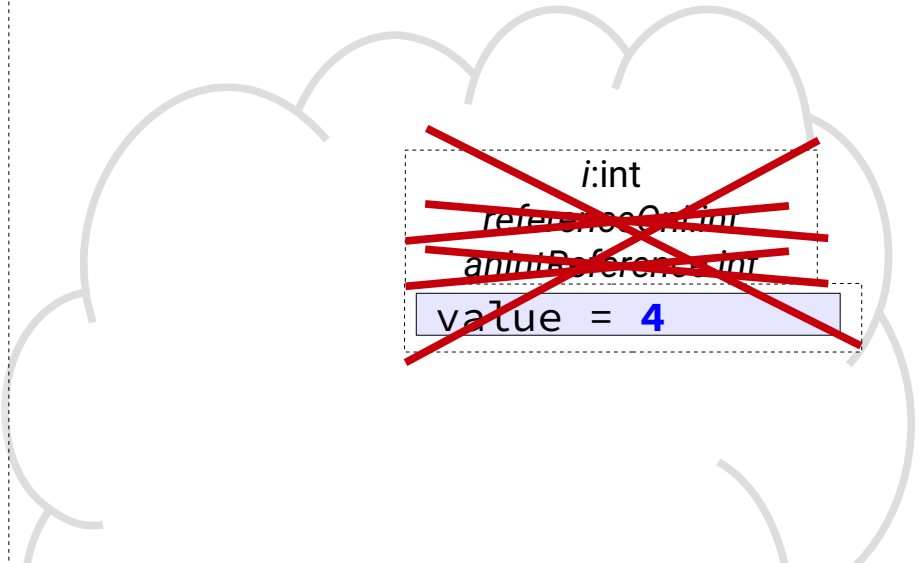
## *passage par référence*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}
```

```
main(){  
    int i;  
    i=3;  
    int& reference0ni = i;  
    incremente(i);  
    std::cout << i <<std::endl;  
}
```

```
jdeanton@FARCI:$/executable  
4  
jdeanton@FARCI:$
```



~~i:int~~  
~~reference0ni:int~~  
~~anIntReference:int~~  
value = 4

Les variables/objets dont la mémoire est allouée **statiquement** sont détruits à la fin du bloc de déclaration

# Memory concerns...

## *passage par référence constante*

- Seule la définition et la déclaration d'une fonction porte la marque de l'utilisation du passage par référence

```
void incremente(const int& anIntReference){  
    anIntReference = anIntReference + 1;  
    return;  
}
```

```
main  
int  
i=3  
int& reference0ni = i;  
incremente(i);  
std::cout << i <<std::endl;  
}
```

i:int  
reference0ni:int

```
jdeanton@linux-ty8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe  
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11  
small_test.cpp: In function 'void incremente(const int&)':  
small_test.cpp:6:18: error: assignment of read-only reference 'anIntReference'  
    6 |         anIntReference = anIntReference + 1;  
      |         ^~~~~~
```

```
jdeanton@FARCI:./executable  
4  
jdeanton@FARCI:$
```

# Memory concerns...

## *à votre avis ?*

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    std::cout << *ptrRes <<std::endl;
}
```

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    std::cout << *ptrRes <<std::endl;
}
```

Dépend de pas mal de chose mais dans tous les cas,  
renvoyer un moyen d'accéder à une variable local, c'est le mal

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
small_test.cpp: In function 'int* incremente(int)':
small_test.cpp:23:10: warning: address of local variable 'anInt' returned [-Wreturn-local-addr]
   23 |     return &anInt;
      |           ^~~~~~
small_test.cpp:20:22: note: declared here
   20 | int * incremente(int anInt){
      |           ~~~~~^~~~~~
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    std::cout << *ptrRes <<std::endl;
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
small_test.cpp: In function 'int* incremente(int)':
small_test.cpp:23:10: warning: address of local variable 'anInt' returned [-Wreturn-local-addr]
   23 |     return &anInt;
      |           ^~~~~~
small_test.cpp:20:22: note: declared here
   20 |     int * incremente(int anInt){
      |                   ~~~~~^~~~~~
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
Segmentation fault (core dumped)
```

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){  
    anInt += 1;  
    return &anInt +1;  
}  
  
main(){  
    int i;  
    i=3;  
    int* ptrRes = incremente(i);  
    std::cout << *ptrRes <<std::endl;  
}
```

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){  
    anInt += 1;  
    return &anInt +1;  
}  
  
main(){  
    int i;  
    i=3;  
    int* ptrRes = incremente(i);  
    std::cout << *ptrRes <<std::endl;  
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe  
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11  
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

**Il n'y a plus de warning !!**



# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt +1;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    std::cout << *ptrRes <<std::endl;
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

**Il n'y a plus de warning !!**

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
-1558645920
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
1794023024
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
1088566016
```

**Il n'y a plus d'erreurs à l'exécution !!!**

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){  
    anInt += 1;  
    return &anInt +1;  
}  
  
main(){  
    int i;  
    i=3;  
    int* ptrRes = incremente(i);  
    *ptrRes = 42 ;  
    std::cout << *ptrRes <<std::endl;  
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe  
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11  
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

**Il n'y a plus de warning !!**

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe  
42
```

**Il n'y a plus d'erreurs à l'exécution !!! On peut même utilisé la mémoire pointée**

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt +1;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    *ptrRes = 42 ;
    std::cout << *ptrRes <<std::endl;
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

**Il n'y a plus de warning !!**

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
42
```

**Il n'y a plus d'erreurs à l'exécution !!! On peut même utilisé la mémoire pointée**

**Et pourtant c'est vraiment faux !**

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
int* incremente(int anInt){
    anInt += 1;
    return &anInt + 200;
}

main(){
    int i;
    i=3;
    int* ptrRes = incremente(i);
    *ptrRes = 42 ;
    std::cout << *ptrRes <<std::endl;
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> make -f Makefile_v1 small_test.exe
g++ -g -c small_test.cpp -o small_test.o -I -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
g++ small_test.o myUtils.o -I -L -o small_test.exe -g -ansi -Wall -Wextra -Wold-style-cast -Woverloaded-virtual -D_DEBUG_ -std=c++11
```

**Il n'y a plus de warning !!**

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe
42
```

**Il n'y a plus d'erreurs à l'exécution !!! On peut même utilisé la mémoire pointée**

**Et pourtant c'est vraiment faux !**

# Memory concerns...

## à votre avis ?

- Quel est le comportement de ce code:

```
main(){  
    int i=3, j = 2;  
    *(&i+ (&j - &i)) = 42;  
    std::cout << i << "----" << j <<std::endl;  
}
```

```
jdeanton@linux-iy8j:~/boulot/enseignements/CPP/2019/cours/example_cours/divers> ./small_test.exe  
3----42
```

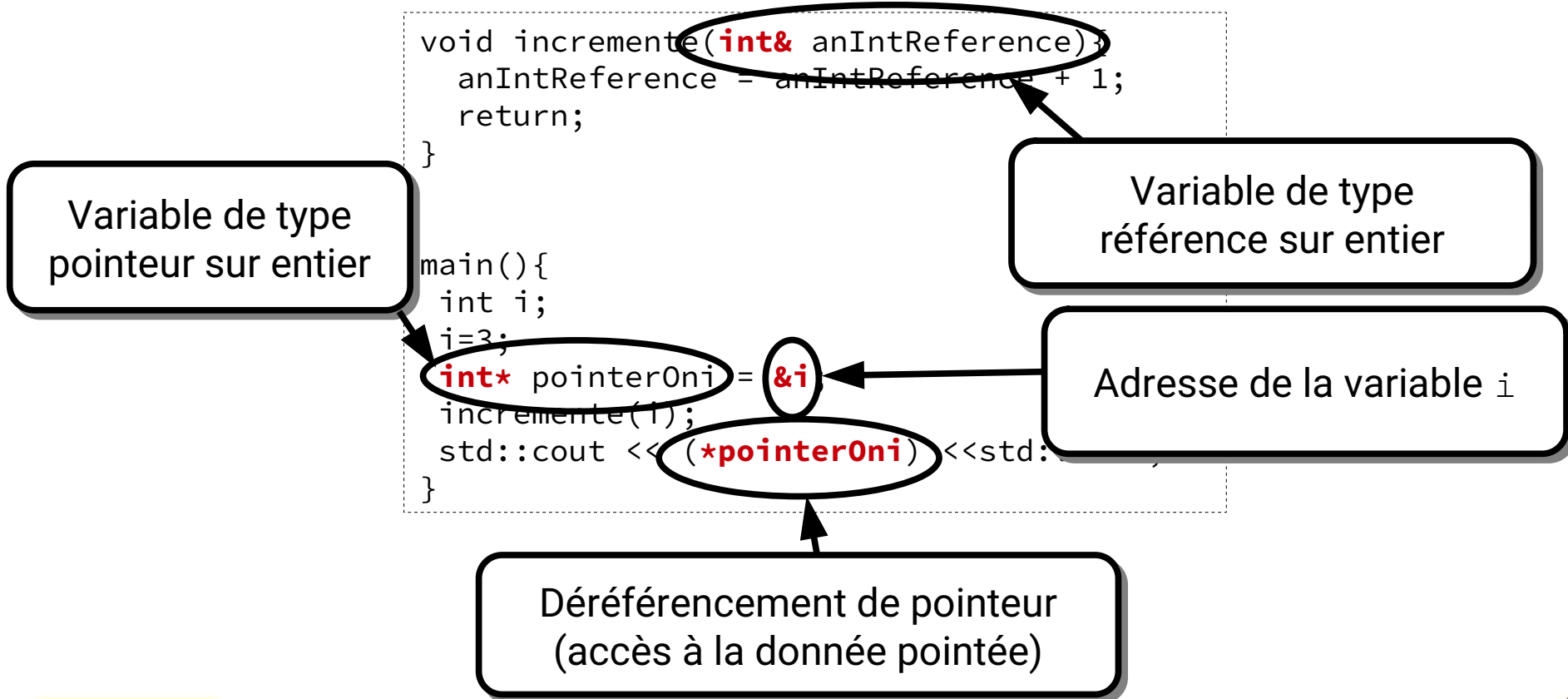
Bref....

**Nous n'allons pas jouer à cela. Donc pas de tableau « à la C » dans ce cours et utilisation MINIMUM des pointeurs !**

# Memory concerns...

## *résumé des notations*

- À connaître par coeur !



# Quelques constructions/mots clefs utiles

- *using* a deux différentes utilités
  - Naviguer/importer d'un workspace
    - *using std::cout*
    - *using namespace XXX*
  - Déclarer de nouveau types
    - *using Row = vector<bool>;*
    - *using Matrix = vector<Row>;*
    - *Template <typename T>*  
*using Matrix<T> = vector<vector<T>>*

# Quelques constructions/mots clefs utiles

- ***inline*** demande au compilateur « d'éviter le cout d'appel à la fonction »
- Le compilateur n'est pas obligé de le faire si il pense que ce n'est pas possible/raisonnable
- De fait, la taille de l'exécutable grossi....

Faites le(s) test(s) vous même !!