Programmation Orienté Objet

LIV 2025-2026

Travaux Dirigés approndissement du C++

Site du cours : `https://defelice.up8.site/poo.html`

`g++ mon_fichier.cpp -std=c++20 -Wall -Wextra -fsanitize=address -o mon_programme`

Remarque :

- Ne pas s'aider de machine sauf pour confirmer.
- Les morceaux de codes ne correspondent pas à des situations pratiques, ils sont là pour illustrer les mécanismes du C++.
- La mise en page du code (saut de ligne, indentation) laisse à désirer le but est d'économiser de la place.
- Il n'y a aucune erreur de syntaxe pure : pas de parenthèse/accolade/point virgule manquant(e) etc ... (ou plutôt : s'il y en a elles ne sont pas intentionnelles et donc généralement non pertinente à relever).
- Certaines réponses sont transparentes, selectionnez-les avec la souris. **Attention :** La réponse est normalement précédée du nom de l'exercice précédent. Si ce n'est pas le cas cela signifie que la réponse n'a pas été vérifiée.

This ThI

Cours 1.

Le mot clé `this` ne s'utilise qu'à l'intérieur d'une méthode d'objet. Il représente l'adresse de l'objet. En particulier si il est utilisé dans une classe `A` alors son type est `A*`.

Exercice. *This ThI1*

Qu'affiche l'exécution ce code ?

```cpp
#include<iostream>
struct A
{
   A* retThis(void) { return this;}
};
int main()
{
   A obj;
   if(obj.retThis()==&obj)
   {
      std::cout << "this est l'adresse de obj" << std::endl;
   }
   else
   {
      std::cout << "this n'est pas l'adresse de obj" << std::endl;
   }
   return 0;
}
```

Réponse : ThI1 □

Exercice. *This ThI6*

Ce code compile. Qu'affiche-t-il à l'execution ?

```cpp
#include<iostream>
struct A
{
   int a;
   void methode()
   {
```

```
7         (*this).a=1;
8     }
9     void methode1()
10    {
11        this->methode();
12        std::cout << a << std::endl;
13    }
14 };
15 int main()
16 {
17    A obj;
18    obj.methode1();
19    return 0;
20 }
```

Réponse : ThI6 □

Exercice. *This ThI2*

Ce code ne compile pas pourquoi ?

```
1  #include<iostream>
2  struct A
3  {
4     int a;
5  };
6  int main()
7  {
8     A obj;
9     A* ptr=this;
10    return 0;
11 }
```

Réponse : ThI2

□

Exercice. *This ThI3*

Ce code ne compile pas. Pourquoi ?

```
1  #include<iostream>
2  struct A
3  {
4     int a;
5  };
6  int main()
7  {
8     A obj;
9     obj.this;
10    return 0;
11 }
```

Réponse : TI3 □

Exercice. *This ThI4*

Qu'affiche ce code ?

```
1  #include<iostream>
2  void methode1(void) { std::cout <<"1F"<<std::endl; }
3  struct A
4  {
5     int a;
6     void methode1(void) { std::cout <<"1M"<<std::endl; }
7     void methode2(void) { std::cout <<"2M"<<std::endl; }
8     void methode3(void) {
```

```
9         this->methode2();
10        methode1();
11    }
12 };
13 int main()
14 {
15    A obj;
16    obj.methode3();
17    return 0;
18 }
```

Réponse : ThI4

□

Exercice. *This ThI5*

Qu'affiche l'exécution ce code ?

```
1  #include<iostream>
2  void methode1(void) { std::cout << "1F"<<std::endl; }
3  struct A
4  {
5     int a;
6     void methode1(void) { std::cout << "1M"<<std::endl; }
7     void methode2(void) { std::cout << "2M"<<std::endl; }
8     void methode3(void) {
9        void methode1(void);
10       this->methode2();
11       methode1();
12    }
13 };
14 int main()
15 {
16    A obj;
17    obj.methode3();
18    return 0;
19 }
```

Réponse : ThI5

□

Membre MI

Exercice. *membres MI2*

Pouquoi ce code ne compile pas ?

```
1  #include<iostream>
2  struct A
3  {
4     int a;
5     void methode(void)
6     {
7        a=1;
8     }
9     void methode3(void)
10    {
11       methode();
12    }
13 };
14 int main()
15 {
16    A obj;
```

```
17      obj.methode3();
18      methode();
19      std::cout << obj.a << std::endl;
20      return 0;
21  }
```

Réponse : MI2

□

Exercice. *membres MI3*

Qu'affiche l'exécution de ce code ?

```
1  #include<iostream>
2  struct A
3  {
4      int a;
5      void setMet(int t)
6      {
7          this->a=t;
8      }
9  };
10 int main()
11 {
12     A x;
13     A y;
14     x.setMet(1);
15     y.setMet(2);
16     std::cout << x.a << std::endl;
17     std::cout << y.a << std::endl;
18     return 0;
19 }
```

Réponse : MI3

□

Exercice. *membres MI1*

Qu'affiche l'exécution de ce code ?

```
1  #include<iostream>
2  struct A
3  {
4      int a;
5      void methode(void)
6      {
7          a=1;
8      }
9      void setMet(int a)
10     {
11         a=5;
12     }
13 };
14 int main()
15 {
16     A obj;
17     obj.a=0;
18     std::cout << obj.a << std::endl;
19     obj.methode();
20     std::cout << obj.a << std::endl;
21     obj.setMet(5);
22     std::cout << obj.a << std::endl; // piege
23     return 0;
24 }
```

Réponse : MI1

□

Publique/Privé PPI

Un membre de classe est soit `private` soit `public` (soit `protected`). Un membre privé d'une classe A ne peut être utilisé que dans une méthode de la classe A.

Exercice. *publique/privé PPI1*

Ce code compile-t-il ? si non pourquoi ?

```
1  struct A{
2     private:
3        int atr;
4  };
5  int main(void){
6     A ob;
7     ob.atr=1;
8     return 0;
9  }
```

Réponse : PPI1

□

Exercice. *publique/privé PPI3*

Ce code compile-t-il ? si non pourquoi ?

```
1  class A{
2     public:
3        int atr;
4  };
5  int main(void)
6  {
7     A ob;
8     ob.atr=1;
9     return 0;
10 }
```

Réponse : PPI3 □

Exercice. *publique/privé PPI5*

Ce code compile-t-il ? si non pourquoi ?

```
1  class A{
2     private :
3        int atr;
4     public:
5        int somme(A obj)
6        {
7           return obj.atr+this->atr;
8        }
9  };
10 int main(void){
11    return 0;
12 }
```

Réponse : □

Exercice. *publique/privé PPI4*

Ce code compile-t-il ? si non pourquoi ?

```
1  class A{
2     int atr;
3  };
4  int main(void){
5     A ob;
```

```
6     ob.atr=1;
7     return 0;
8  }
```

Réponse : □

Exercice. *publique/privé PPI2*

1. Ce code compile-t-il ? si non pourquoi ?

```
1  class A{
2     int atr;
3  };
4  int main(void){
5     A ob;
6     return 0;
7  }
```

Réponse : PPI2.1 □

2. Ce code compile-t-il ? si non pourquoi ?

```
1  class A{
2     A(void){}
3     int atr;
4  };
5  int main(void)
6  {
7     A ob; // piege
8     return 0;
9  }
```

Réponse : PPI2.2 □

Surcharge I, SI

On peut déclarer (ou définir) dans un même espace de nom plusieurs fonctions (ou méthodes) ayant le même identifiant à la condition qu'elles aient des signatures différentes 2 à 2. Dans un tel cas, on dit que "la" fonction est surchargée. Lors d'un appel à une fonction (ou une méthode) surchargée le compilateur choisie la fonction ayant "la meilleure" correspondance de type entre les paramètres et les arguments (voir plus bas). Si il n'y a pas de "meilleure" candidate, alors la compilation produit une erreur.

Si l'appel explicite un seul argument, la meilleure fonction est celle qui possède un seul paramètre ayant le même type que l'argument. S'il y en a pas alors le compilateur cherche à convertir l'argument dans le type du paramètre en utilisant une conversion implicite. Il existe des conversions meilleures que d'autres mais nous ne préciseront pas lesquelles ici. Si le compilateur trouve plusieurs candidates ou aucune alors il produit une erreur

Dans le cas d'un appel à plusieurs arguments la "meilleure" candidate est déterminée ainsi : pour chaque argument on regarde les meilleurs candidates (voir choix à un paramètre au dessus). Ce qui produit un ensemble de meilleurs candidat par argument d'appel. Si l'intersection de ces ensembles contient un seul candidat alors il est choisi, sinon (pas de candidat ou plusieurs) le compilateur produit une erreur.

Le cas des valeurs par défaut des paramètre produit différentes signatures.

Exercice. *surcharge SI5*

Ce code produit-il une erreur à la compilation ?

```
1  #include<iostream>
2  void f(int truc) {}
3  void f(char a) {}
4  int main(void)
5  {
6     return 0;
7  }
```

Réponse : SI5 □

Exercice. *surcharge SI3*

Ce code produit une erreur à la compilation, pourquoi ?

```
1  #include<iostream>
2  void f(char truc) {}
3  int* f(char a) {return nullptr;}
```

```
4  int main(void)
5  {
6      return 0;
7  }
```

Réponse : SI3 □

Exercice. *surcharge SI1*

Qu'affiche ce code ?

```
1  #include<iostream>
2  void f(int a) { std::cout << "je suis f1 " << std::endl; }
3  void f(long  a) { std::cout << "je suis f2 " << std::endl; }
4  void f(void) { std::cout << "je suis f3 " << std::endl; }
5  void f(char b,long d) { std::cout << "je suis f4 " << std::endl; }
6  int main(void)
7  {
8      f();
9      f(1l);
10     f(1l,'a'); // c'est bien f(1l,'a') et c'est pas f('a',1l)
11     f(3);
12     return 0;
13 }
```

Exercice. *surcharge SI2*

Ce code produit une erreur à la compilation, pourquoi ?

```
1  #include<iostream>
2  void f(char a) { std::cout << "je suis f1 " << std::endl; }
3  int* f(int a) { std::cout << "je suis f2 " << std::endl; return nullptr; }
4  int main(void)
5  {
6      f(1l);
7      return 0;
8  }
```

Réponse : SI2

□

Exercice. *surcharge SI6*

Ce code compile. Qu'affiche son exécution ? Que se passe-t-il si on décommente la dernière ligne ?

```
1  #include<iostream>
2  void f(char a,double b,long c ) { std::cout << "je suis f1" << std::endl; }
3  void f(int a,char b,double c) { std::cout << "je suis f2" << std::endl; }
4  void f(long a,double b,char c) { std::cout << "je suis f3" << std::endl; }
5  int main(void)
6  {
7      double d=4.;
8      char c=3;
9      long l=1;
10     int i=1;
11
12      f(i,l,d);
13      f(d,l,d);
14      f(c,i,i);
15     // f(c,c,c);
16     return 0;
17 }
```

Réponse : SI6 □

Exercice. *surcharge SI7*

Ce code compile-t-il ? Si oui qu'affiche-t-il ?

```
#include<iostream>
void f(char a,double b,char c ) { std::cout << "je suis f1" << std::endl; }
void f(int a,char b,double c) { std::cout << "je suis f2" << std::endl; }
int main(void)
{
    double d=4.;
    char c=3;
    long l=1;
    int i=1;

    f(c,l,c);
    return 0;
}
```

Réponse : SI6 □

Exercice. *surcharge SI4*

Ce code compile correctement. Qu'affiche-t-il ?

```
#include<iostream>
struct A
{
    A(){
        std::cout<<"je suis A()"<<std::endl;
    }
    A(int b){
        std::cout<<"je suis A(int) avec "<< b << std::endl;
    }
};
int main(void)
{
    A a;
    A b{5};
    A c={3};
    A d(2);
    A f{};
    A e(); // piege
    return 0;
}
```

Construction/Destruction CDI

Exercice. *construction/destruction CDI2*

Qu'affiche ce programme à l'exécution ?

```
#include<iostream>
class A{
    int atr;
    public:
    A(void) { std::cout << "Construction"  << std::endl; }
    ~A(void) { std::cout << "Destruction" << std::endl; }
};
int main(void)
{
    A ob;
    return 0;
}
```

Réponse : □

Exercice. *construction/destruction CDI8*

Qu'affiche ce programme à l'exécution ?

```
1  #include<iostream>
2  struct A{
3     A(void) { std::cout << "Construction"  << std::endl; }
4     ~A(void) { std::cout << "Destruction" << std::endl; }
5  };
6  int main(void)
7  {
8     A ob1;
9     A* ob3;
10    std::cout << "ici" << std::endl;
11    A ob2{};
12    A& ob4=ob1;
13    return 0;
14 }
```

Réponse : CDI8 □

Exercice. *construction/destruction CDI3*

Qu'affiche ce programme à l'exécution ?

```
1  #include<iostream>
2  class A{
3     int atr;
4     public:
5     A(int a)
6     {
7        atr=a;
8        std::cout << "C" << atr << "␣";
9     }
10    ~A(void) { std::cout << "D"  << atr << "␣"; }
11 };
12 int main(void)
13 {
14    A ob0{0};
15    A ob1=1;
16    {
17       A ob2(2);
18       A ob3={3};
19    }
20    A ob4{4};
21    return 0;
22 }
```

Réponse : □

Exercice. *construction/destruction CDI7*

Ce code produit une erreur à la compilation. Pourquoi ?

```
1  class A{
2     int a;
3     A(){}
4     A(int u) {
5        A b;
6        b.a=u;
7        return b;
8     }
9  };
10 int main(void)
```

```
11 {
12     return 0;
13 }
```

Réponse : CDI7 □

Exercice. *construction/destruction CDI11*

Le code suivant produit une erreur. Pourquoi ?

```
1  #include<iostream>
2  class A{
3      public:
4      A(int a){
5          std::cout << "Construction" ;
6      }
7  };
8  int main(void){
9      A a;
10     return 0;
11 }
```

Réponse : □

Le code suivant produit-t-il une erreur ?

```
1  #include<iostream>
2  class A{
3  };
4  int main(void){
5      A a;
6      return 0;
7  }
```

Réponse : □

Exercice. *construction/destruction CDI5*

Qu'affiche ce code ? Que se passe si on décommente la ligne commentée ?

```
1  #include<iostream>
2  struct A{
3      A(void){
4          std::cout << "Construction A1" << std::endl;}
5      A(long a){
6          std::cout << "Construction A2" << std::endl;}
7      A(char a){
8          std::cout << "Construction A3" << std::endl;}
9      A(float a,int b=2){
10         std::cout << "Construction A4" << std::endl;}
11 };
12 int main(void){
13     A ob0;
14     A ob1{};
15     A ob2{3l};
16     A ob3(3.f);
17     A ob4{3,2};
18     //A ob5{3};// (que se passe-t-il si on decommente la ligne ?)
19     return 0;
20 }
```

Réponse : □

Exercice. *construction/destruction CDI10 (anciennement CDI9)*

Qu'affiche ce programme à l'exécution ?

```
1  #include<iostream>
2  struct A{
3      A(void) { std::cout << "Construction" << std::endl; }
4      ~A(void) { std::cout << "Destruction" << std::endl; }
5  };
6  int main(void)
7  {
8      A ob1;
9      ob1.~A();
10     std::cout<<"ici"<<std::endl;
11     return 0;
12 }
```

Réponse : CDI10 □

Référence RI

On peut voir une référence (ici lvalue référence : `A&`) comme un pointeur constant qui s'utilise comme une variable classique. Lorsque le type de retour d'une fonction est une référence alors l'expression de l'appel de la fonction est considérée comme une lvalue. (lvalue : (grossièrement) expression que l'on peut mettre à gauche d'une affectation, qui représente un emplacement dans la mémoire)

Exercice. *Référence RI0*

Ce code compile. Qu'affiche son exécution ?

```
1  #include<iostream>
2  int main()
3  {
4      int c=0;
5      int& b=c;
6      std::cout << c << std::endl ;
7      std::cout << b << std::endl ;
8      c=1;
9      std::cout << b << std::endl ;
10     std::cout << c << std::endl ;
11     b=2;
12     std::cout << b << std::endl ;
13     std::cout << c << std::endl ;
14     return 0;
15 }
```

Réponse : Réponse RI0 □

Exercice. *Référence RI8*

Ce code ne compile pas. Pourquoi ?

```
1  int main() {
2      int a;
3      int& b;
4      b=a;
5      return 0;
6  }
```

Réponse : □

Exercice. *Référence RI3*

Ce code ne compile pas. Pourquoi ?

```
1  int main() {
2      int& b=1;
3      return 0;
4  }
```

Réponse : □

Exercice. *Référence RI1*

Ce code compile. Qu'affiche-t-il ?

```
1  #include<iostream>
2  void f(int a) {
3      a=3;
4  }
5  void g(int& b) {
6      b=2;
```

```
}
int main() {
    int c=0;
    f(c);
    std::cout << c << std::endl ;
    g(c);
    std::cout << c << std::endl ;
    return 0;
}
```

Réponse : □

Exercice. *Référence RI7*

Ici y a t-il le même problème ? Qu'affiche l'exécution ? Quels changements se produisent si on décommente la ligne ?

```
#include<iostream>
int& f(int& a)
{
    return a;
}
int g(int& a)
{
    return a;
}
int main()
{
    int c=1;
    std::cout << c << std::endl ;
    f(c)=6;
    // g(c)=7;
    std::cout << c << std::endl ;
    return 0;
}
```

Réponse : RI7 2

□

Ce code compile mais quel est le problème à l'exécution ?

```
int& f(int a)
{
    return a;
}
int main()
{
    int c=1;
    f(c)=6;
    return 0;
}
```

Réponse : RI7 1

□

Exercice. *Référence RI9*

Ce code ne compile pas pourquoi ?

```
int& f(void)
{
    return 3;
}
int main()
{
    return 0;
}
```

Réponse : □

Exercice. *Référence RI6*

Ce code compile sans erreur. Qu'affiche son exécution ?

```
#include<iostream>
struct A{
    A() { std::cout << "A()" << std::endl; }
};
int main()
{
    A a;
```

```
8     A& b=a;
9     return 0;
10 }
```

Réponse : □

Objet Membre OMI

Exercice. *OMI0*

Ce code compile et s'exécute correctement. Qu'affiche son exécution ?

```
1  #include<iostream>
2  struct X{
3     X(){ std::cout << "CX ";}
4     ~X(){std::cout << "DX ";}};
5  struct Y{
6     Y(){ std::cout << "CY ";}
7     ~Y(){std::cout << "DY ";}};
8  struct Z{
9     X atrX;
10    Y atrY;
11    Z(void){ std::cout << "CZ "; }
12    ~Z(void){ std::cout << "DZ "; }};
13 int main(void){
14    Z a;
15    std::cout << "ici ";
16    return 0;
17 }
```

Réponse : OMI0 □

Exercice. *OMI1*

Ce code compile et s'exécute correctement. Qu'affiche son exécution ?

```
1  #include<iostream>
2  struct X{
3     X(){ std::cout << "CX ";}
4     ~X(){std::cout << "DX ";}};
5  struct Y{
6     X chmpX;
7     Y(){ std::cout << "CY ";}
8     ~Y(){std::cout << "DY ";}};
9  struct Z{
10    Y atrY;
11    X atrX;
12    Z(void){ std::cout << "CZ "; }
13    ~Z(void){ std::cout << "DZ "; }};
14 int main(void){
15    Z a;
16    std::cout << "ici ";
17    return 0;
18 }
```

Réponse : OMI1 □

Exercice. *OMI2*

La compilation de ce code produit une erreur, pourquoi ?

```
1  #include<iostream>
2  struct X{
3     X(int a){}
```

```
};
struct Y{
   X atrX;
};
int main(void){
   Y y;
   return 0;
}
```

Réponse : OMI2-1□

Exercice. *OMI3*

Qu'affiche ce code ? Que se passe-t-il si on décommente la ligne commentée ?

```
#include<iostream>
struct X{
   X(int a){std::cout<<"CXi ";}
   X(long a){std::cout<<"CXl ";}
};
struct Y{
   X atrX1;
   X atrX2{2};
   Y(): atrX1{2},atrX2{3l}{}
   Y(int p) : atrX1{3l} {}
   //Y(int p,int b) :atrX2{1} {}
};
int main(void){
   Y a;
   Y b{1};
   return 0;
}
```

Réponse : OMI3 □

Exercice. *OMI4*

Quel est le (léger) problème avec ce code ? (il compile mais ...)

```
#include<iostream>
struct X{ };
struct Y{
   X atrX1;
   X atrX2{};
   Y(): atrX2{}, atrX1{}{}
};
int main(void){
   return 0;
}
```

Réponse : OMI4 □

Exercice. *OMI5*

La compilation de ce code produit une erreur, pourquoi ?

```
struct X{
   int& a;
};
int main(void){
   X b;
   return 0;
}
```

Réponse : OMI5 □

New, delete NDI

Exercice. *new/delete NDI1*

Qu'affiche l'execution du code suivant ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "C" << p <<  std::endl;
7      }
8      ~A(){
9          std::cout << "D" << a << std::endl;
10     }
11 };
12 int main(void){
13     A* p=new A{5};
14     A a(1);
15     delete p;
16     return 0;
17 }
```

Réponse : □

Exercice. *new/delete NDI6*

Qu'affiche l'execution du code suivant ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "C" << p <<  std::endl;
7      }
8      ~A(){
9          std::cout << "D" << a << std::endl;
10     }
11 };
12 int main(void){
13     A* p=(A*)malloc(sizeof(A));
14     free(p);
15     return 0;
16 }
```

Réponse : □

Exercice. *new/delete NDI2*

Qu'affiche l'execution code suivant ? Quel est le problème ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "C" << p <<  std::endl;
7      }
8      ~A(){
9          std::cout << "D" << a << std::endl;
10     }
11 };
12 int main(void){
```

```
13      A* p=new A{5};
14      A a(1);
15      return 0;
16  }
```

Réponse : □

Exercice. *new/delete NDI3*

Qu'affiche l'execution du code suivant ? Y-a t-il une fuite de mémoire ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "C" << p <<  std::endl;
7      }
8      ~A(){
9          std::cout << "D" << a << std::endl;
10     }
11 };
12 int main(void){
13     A a{2};
14     new(&a)A{4};
15     return 0;
16 }
```

Réponse : □

Exercice. *new/delete NDI4*

```
1  #include<iostream>
2  class A
3  {
4      int atr;
5      public:
6      A(void)
7      {
8          std::cout << "Bonjour"  << std::endl;
9      }
10     ~A(void)
11     {
12         std::cout << "Au revoir"  << std::endl;
13     }
14 };
15 int main(void)
16 {
17     A ob;
18     A* obj2= new A();
19     A* obj3;
20     {
21         A obj4;
22         delete obj2;
23         obj2= new A();
24     }
25     A obj5;
26     return 0;
27 }
```

Réponse : □

Exercice. *new/delete NDI5*

Ce code compile. En supposant que tout se passe bien, présente-t-il une fuite de mémoire à l'exécution ?

```
#include<iostream>
#include<cstdlib>
class A { };
int main(void)
{
    int i;
    A* ptr=(A*)malloc(sizeof(A)*10);
    if(ptr==nullptr)
    {
        exit(1);
    }
    for(i=0;i<5;i++)
    {
        new(ptr+i)A();
    }
    free(ptr);
    return 0;
}
```

Réponse : □

Exercice. *new/delete NDI7*

Ce code compile. En supposant que tout se passe bien, présente-t-il une fuite de mémoire à l'exécution ?

```
#include<iostream>
#include<cstdlib>
class A {
    int* p=new int[10];
};
int main(void)
{
    int i;
    A* ptr=(A*)malloc(sizeof(A)*10);
    if(ptr==nullptr)
    {
        exit(1);
    }
    for(i=0;i<5;i++)
    {
        new(ptr+i)A();
    }
    free(ptr);
    return 0;
}
```

Réponse : □

Référence 2 R2I

Exercice. *Référence 2 R2I2*

La compilation de ce code ne produit pas d'erreur. Qu'affiche son exécution ? Que se passe-t-il si on décommente la ligne.

```
#include<iostream>
int main()
{
    //int &  a=3;
    const int & b=2;
    return 0;
}
```

Réponse : Réponse RI0 □

Exercice. *Référence 2 R2I3*

Ce code produit une erreur à la compilation. Quelle est la ligne qui pose problème et pourquoi ?

```
1  #include<iostream>
2  int main()
3  {
4      const int & b=2;
5      b=1;
6      return 0;
7  }
```

Réponse : Réponse RI0 □

Tableau TI

Exercice. *tableau TI9*

Ce code compile. Qu'affiche ce code ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(){
5          std::cout << "C" <<  std::endl;
6      }
7      ~A(){
8          std::cout << "D" << std::endl;
9      }
10 };
11 int main(void)
12 {
13     A T[4];
14     std::cout << "ici" << std::endl;
15     return 0;
16 }
```

Réponse : □

Exercice. *tableau TI2*

Ce code ne compile pas. Pourquoi ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "C" <<  std::endl;
7      }
8      ~A(){
9          std::cout << "D" << std::endl;
10     }
11 };
12 int main(void)
13 {
14     A T[4]; // ici erreur
15     return 0;
16 }
```

Réponse : □

Exercice. *tableau TI8*

Quel message affiche ce code ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(int p){
5          a=p;
6          std::cout << "Ci" <<  std::endl;
7      }
8      A(){
9          std::cout << "C" <<  std::endl;
10     }
11     ~A(){
12         std::cout << "D" << std::endl;
13     }
14 };
15 int main(void)
16 {
17     A T[]{1,{},{3},1,9};
18     std::cout << "ici" << std::endl;
19     return 0;
20 }
```

Réponse : TI5 □

Exercice. *tableau TI4*

Pourquoi ce code produit-il une erreur à la compilation ?

```
1  #include<iostream>
2  struct A{
3      A(int a){}
4  };
5  int main(void){
6      A* a;
7      a=new A[20];
8      return 0;
9  }
```

Réponse : □

Exercice. *tableau TI3*

Ce code compile mais quel est son problème ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      A(){
5          std::cout << "C" <<  std::endl;
6      }
7      ~A(){
8          std::cout << "D" << std::endl;
9      }
10 };
11 int main(void)
12 {
13     A* ptr=new A[10];
14     delete ptr;
15     return 0;
16 }
```

Réponse : TI3 □

Exercice. *tableau TI7*

Qu'affiche l'exécution de ce code ?

```
#include<iostream>
struct A{
    int a;
    A(){ std::cout << "C" <<  std::endl; }
    A(int i){ std::cout << "Ci" <<  std::endl; }
    ~A(){
        std::cout << "D" << std::endl;
    }
};
int main(void)
{
    A* ptr=new A[3]{{},{1},{2}};
    std::cout << "ici" << std::endl;
    delete[] ptr;
    std::cout << "la" << std::endl;
    return 0;
}
```

Réponse : TI6 □

Classe const/non const CCNCI

Exercice. *const/non const CCNCI8*

Ce code produit une erreur de compilation. Quelle est la ligne qui pose problème et pourquoi ?

```
struct A{
    int c=0;
};
int main(void){
    A a1;
    a1.c=3;
    const A a2;
    int u=a2.c;
    a2.c=2;
    return 0;
}
```

Réponse : □

Exercice. *const/non const CCNCI6*

Ce code produit une erreur de compilation. Laquelle ?

```
#include<iostream>
struct A{
    void f(void){
        std::cout << "methode_A::f" << std::endl;
    }
};
int main(void){
    A const obj;
    obj.f();
    return 0;
}
```

Réponse : □

Exercice. *const/non const CCNCI2*

Ce code produit-il une erreur de compilation ?

```cpp
#include<iostream>
struct A{
    void f(void) const {
        std::cout << "methode A::f" << std::endl;}
    };
int main(void){
    A const obj;
    obj.f();
    return 0;
}
```

Réponse : □

Exercice. *const/non const CCNCI1*

Ce code compile-t-il ?

```cpp
struct A{
    int a;
    int methode() { }
    void methode() { }
};
int main(void)
{
    return 0;
}
```

Et celui-ci ?

```cpp
struct A{
    int a;
    void methode() const { }
    void methode() { }
};
int main(void)
{
    return 0;
}
```

Réponse : □

Exercice. *const/non const CCNCI3*

Ce code compile. Qu'affiche son exécution ?

```cpp
#include<iostream>
struct A{
    void methode() const {
        std::cout << "je suis const" << std::endl;
    }
    void methode() {
        std::cout << "je ne suis pas const" << std::endl;
    }
};
int main(void)
{
    A objnc;
    const A objc;
    objc.methode();
    objnc.methode();
    return 0;
}
```

Réponse : □

Exercice. *const/non const CCNCI4*

Ce code compile t-il ? si oui qu'affiche-t-il ? Si non pourquoi ?

```cpp
#include<iostream>
struct A{
    void methode() const {
        std::cout << "je suis const" << std::endl;
    }
};
int main(void)
{
    A objnc;
```

```
10      const A objc;
11
12      objc.methode();
13      objnc.methode();
14
15      return 0;
16  }
```

Réponse : □

Exercice. *const/non const CCNCI5*

Ce code compile t-il ? Si oui, qu'affiche-t-il ? Si non pourquoi ?

```
1  #include<iostream>
2  struct A{
3      int a;
4      void f(void) const {
5          a=3;
6      }
7  };
8  int main(void){
9      return 0;
10 }
```

Réponse : □

Exercice. *const/non const CCNCI7*

Ce code compile t-il ? Si oui, qu'affiche-t-il ? Si non pourquoi ?

```
1  #include<iostream>
2  struct A{
3      mutable int a;
4      void f(void) const {
5          a=3;
6      }
7  };
8  int main(void){
9      return 0;
10 }
```

Réponse : □

Membre de classe MCI

Exercice. *membre de classe MCI6*

Ce code compile, qu'affiche-t-il en s'exécutant ?

```
1  #include<iostream>
2  struct A{
3      static int atS;
4      int at;
5      void affichage() {
6          std::cout << atS << std::endl;
7          std::cout << at  << std::endl;
8      }
9      A(){
10         at=0;
11     }
12 };
13 int A::atS=0;
```

```
14
15 int main()
16 {
17     A b,a;
18     a.affichage();
19     b.atS=1;
20     b.at=1;
21     a.affichage();
22     A::atS=2;
23     std::cout << A::atS << std::endl;
24     return 0;
25 }
```

Réponse : □

Exercice. *membre de classe MCI7*

Ce code ne compile pas. Pourquoi ?

```
1  #include<iostream>
2  struct A{
3      static int atS;
4      int at;
5      A(){
6          at=0;
7      }
8  };
9  int A::atS=0;
10
11 int main()
12 {
13     A b,a;
14     std::cout << A::at << std::endl;
15 }
```

Réponse : □

Exercice. *membre de classe MCI10*

La commande suivante provoque une erreur à l'édition de lien. Pourquoi ?

```
xxxx@xxxx$ g++ main.cpp maClasse.cpp
```

Listing 1 – maClasse.hpp

```
1  #ifndef _maClasse_HPP_
2  #define _maClasse_HPP_
3  struct maClasse_t{
4      static int atS;
5      int at;
6      void afficher(void);
7  };
8  int maClasse_t::atS=0; //ici
9  #endif
```

Listing 2 – maClasse.cpp

```
1  #include<iostream>
2  #include"maClasse.hpp"
3  using namespace std;
4  void maClasse_t::afficher()
5  {
6      cout<<"affichage()"<<endl;
7  }
```

Listing 3 – main.cpp

```
1  #include"maClasse.hpp"
2  int main()
3  {
4      maClasse_t a;
5      a.afficher();
6  }
```

Réponse : □

Exercice. *membre de classe MCI5*

Ce code ne compile pas. Pourquoi ?

```
1  #include<iostream>
2  struct A{
3      static int atS;
4      void affichage() {
5          std::cout << "Bonjour" << std::endl;
6      }
```

```
};
int A::atS=0;

int main()
{
    A b,a;
    a.affichage();
    A::affichage();
}
```

Réponse : □

Exercice. *membre de classe MCI3*

Ce code compile. Qu'affiche son exécution ?

```
#include<iostream>
struct A{
    static int atS;
    int at;
    A(){
        atS++;
        at=atS;
    }
    static void affichage() {
        std::cout << atS << std::endl;
        atS++;
    }
};
int A::atS=0;

int main()
{
    A b,a;
    A::affichage();
    a.affichage();
}
```

Réponse : □

Exercice. *membre de classe MCI8*

Ce code ne compile pas. Pourquoi ?

```
#include<iostream>
struct A{
    static int atS;
    int at;
    A(){
        atS++;
        at=atS;
    }
    static void affichage() {
        std::cout << atS << at  << std::endl;
        atS++;
    }
};
int A::atS=0;

int main()
{
}
```

Réponse : □

Exercice. *membre de classe MCI9*

Ce code ne compile pas. Pourquoi ?

```
#include<iostream>
struct A{
    static int atS;
    int at;
    A(){
        atS++;
        at=atS;
    }
    void affichage() {
        std::cout << atS << at << std::endl;
        atS++;
    }
    void static affichageStat(){
        affichage();
    }
};
int A::atS=0;

int main() {return 0; }
```

Réponse : □

Conversion CI

En plus des conversions classiques entre types (int vers float par exemple), les constructeurs (s'ils ne sont pas déclarés explicites) engendrent des conversions implicites.

D'autre part, si `A` est un type (non const) alors il existe une conversion implicite de `A*` vers `const A*`. Et puisque le "type" référence vers A (i.e : A&) peut être vu comme une "sorte" de pointeur vers `A`, il existe également une conversion implicite de `A&` vers const `A&`.

Exercice. *Conversion CI1*

Ce code compile, qu'affiche son exécution ?

```
#include<iostream>
struct A{
    A(int a) { std::cout << "A(int) : " << a << std::endl; }
};
void f(A c){}
int main()
{
    A b=5;
    f(1);
    return 0;
}
```

Réponse : □

Exercice. *Conversion CI2*

Ce code compile, qu'affiche son exécution ?

```
#include<iostream>
struct A{
    A(int a)
    {
        std::cout << "A(int) : " << a << std::endl;
    }
};
```

```
8  void f(A c){}
9  A g(){return 3;}
10 int main()
11 {
12     A b=5;
13     double d=2.3;
14     f(1);
15     f(b);
16     f(d);
17     A c=g();
18     return 0;
19 }
```

Réponse : □

Exercice. *Conversion CI3*

Ce code produit une erreur à la compilation, quelles sont les lignes responsables ?

```
1  #include<iostream>
2  struct A{
3      explicit A(int a) { std::cout << "A(int) : " << a << std::endl; }
4  };
5  void f(A c){}
6  int main()
7  {
8      A a{3};
9      //A b=5;
10     f(1);
11     return 0;
12 }
```

Réponse : □

Exercice. *Conversion CI4*

Ce code produit une erreur de compilation.

```
1  int main()
2  {
3      const int a=3;
4      int* b=&a;
5  }
```

Réponse : □

Mais pas celui-ci. Pourquoi ?

```
1  int main()
2  {
3      int a=3;
4      const int* b=&a;
5  }
```

Réponse : □

Exercice. *Conversion CI5*

Ce code ne compile pas. Pourquoi ?

```
1  int main()
2  {
3      const int a=3;
4      int& b=a;
5      return 0;
6  }
```

Réponse : □

Ce code ne produit pas d'erreur de compilation, est-ce vrai ? Qu'affiche son exécution ? Et si on décommente la ligne ?

```
1  #include<iostream>
2  int main()
3  {
4      int a=2;
5      const int& b=a;
6      a=3;
7      //b=1;
8      std::cout << b << std::endl;
9      return 0;
10 }
```

Réponse : □

Surcharge 2 S2I

Pour un type A (qui n'est pas "référence sur" et qui n'est pas "const"), les 2 types de paramètres de A& et const A& engendrent 2 signatures différentes.

Exercice. *Surcharge 2 S2I1*

Ce code compile sans erreur, qu'affiche son exécution ?

```
#include<iostream>
void f(int& a) {
    std::cout << "f1 " ;
}
void f(const int& a) {
    std::cout << "f2 " ;
}
int main()
{
    int a;
    const int b=1;
    f(a);
    f(1);
    f(b);
    return 0;
}
```

Réponse : □

Exercice. *Surcharge 2 S2I2*

Ce code compile-t-il sans erreur ?

```
#include<iostream>
void f(const int& a) {
    std::cout << "f2 " ;
}

int main()
{
    int a;
    const int b=1;
    f(a);
    f(1);
    f(b);
    return 0;
}
```

Réponse : □

Exercice. *Surcharge 2 S2I3*

Ce code compile sans erreur mais si on décommente une seule des lignes commentées le code n'est plus compilable. Le problème est que pour chacune de ces 4 lignes (chacune étant un appel à la fonction f) il y a ambiguité sur la version de f à appeler (ce qui provoque l'erreur). Indiquer pour chacun des 4 appels les candidats f possibles (f1,f2,f3).

```
#include<iostream>
struct A{
    A(int){}
};
void f(const A& a) {
    std::cout << "f2 " ;
}
void f(A& a) {
    std::cout << "f1 " ;
}
void f(A a) {
    std::cout << "f3 " ;
}

int main()
```

```
{
    A a{3};
    const A b{1};
   // f(4); // cas 1
   // f(b); // cas 2
   // f(a); // cas 3
   // f(A{5}); // cas 4
    return 0;
}
```

Réponse : □

Operateur C OCI

Cette section contient des exercices qui portent sur les règles de priorité et d'ordre d'exécution des opérateurs. C'est un rappel du langage C qui est valable en C++.

Exercice. *Operateur C OCI2*

Ce code compile sans erreur, affiche-t-il 2 ou 0 (on rappelle qu'en C++ `2/3` donne 0) ?

```
#include<iostream>
int main()
{
    std::cout << 3*2/3 << std::endl;
    return 0;
}
```

Réponse : OCI2 □

Ce code compile et s'exécute sans erreur, qu'affiche-t-il ? 0 ou 2 ?

```
#include<iostream>
int main()
{
    std::cout << 2/3*3 << std::endl;
    return 0;
}
```

Réponse : OCI2 □

Exercice. *Operateur C OCI3*

Ce code compile sans erreur, affiche-t-il 6 ou 5 (on rappelle qu'en C++ `2/3` donne 0) ?

```
#include<iostream>
int main()
{
    std::cout << 2*2+1 << std::endl;
    return 0;
}
```

Réponse : OCI2 □

Ce code compile et s'exécute sans erreur, qu'affiche-t-il ? 6 ou 5 ?

```
#include<iostream>
int main()
{
    std::cout << 1+2*2 << std::endl;
    return 0;
}
```

Réponse : OCI2 □

Exercice. *Operateur C OCI1*

Ce code compile et s'exécut sans erreur, qu'affiche-t-il ? -1 ou 1 ? (**Attention !** faire un essai avec un compilateur ne permet pas d'avoir bonne réponse !)

```
#include<iostream>
int main()
{
    int i=0;
    std::cout << (i++)-(i++) << std::endl;
    return 0;
}
```

Réponse : OCI1

□

Exercice. *Operateur C OCI4*

Ce code compile et s'exécute sans erreur, qu'affiche-t-il ? -1 ou 1 ? (**Attention !** faire un essai avec un compilateur ne permet pas d'avoir bonne réponse !)

```
1  #include<iostream>
2  int f(int a,int b)
3  {
4      return a-b;
5  }
6  int main()
7  {
8      int i=0;
9      std::cout << f(i++,i++) << std::endl;
10     return 0;
11 }
```

Réponse : OCI4

□

Exercice. *Operateur C OCI5*

Ce code compile et s'exécute sans erreur, qu'affiche-t-il ? 1 ou 2

```
1  #include<iostream>
2  int main()
3  {
4      int i=0;
5      i++ && i++;
6      std::cout << i << std::endl;
7      return 0;
8  }
```

Réponse : OCI5.1

□

Ce code compile et s'exécute sans erreur, qu'affiche-t-il ? 2 ou 3 ?

```
1  #include<iostream>
2  int main()
3  {
4      int i=1;
5      i++ && i++;
6      std::cout << i << std::endl;
7      return 0;
8  }
```

Réponse : OCI5.2

□

Héritage 1 H1I

Exercice. *Héritage 1 H1I1*

Ce code suivant compile sans erreur, qu'affiche-t-il lorsqu'on l'exécute ? Que se passe-t-il si on décommente les lignes commentées ?

```
1  #include<iostream>
2  struct A{
3      int x=0;
4      void methodeA(){
5          std::cout << x << std::endl;
6      }
7  };
8  struct B : public A {
9      int z=1;
10     void methodeB(){
11         std::cout << z << std::endl;
12     }
13 };
14 int main(){
15     A a;
16     B b;
17     a.x=2;
```

```
18      b.x=3;
19      a.methodeA();
20      b.methodeB();
21      b.methodeA();
22      // a.methodeB();
23      // a.z=1
24      return 0;
25  }
```

Réponse : OCI2 □

Exercice. *Héritage 1 H1I6*

La compilation de ce code ne produit pas d'erreur. Qu'affiche son exécution ?

```
1   #include<iostream>
2   struct A{
3      int x;
4   };
5   struct B : public A {
6      int z;
7   };
8   void f(int a)
9   {
10     if(a==0)
11        std::cout << "Faux" << std::endl;
12     else
13        std::cout << "Vrai" << std::endl;
14  }
15  int main(){
16     f( sizeof(A)<=sizeof(B));
17     f( sizeof(A) > sizeof(B));
18     return 0;
19  }
```

Réponse : □

Exercice. *Héritage 1 H1I3*

Ce code produit-il une erreur ? Si oui pourquoi ? Si non qu'affiche-t-il ?

```
1   #include<iostream>
2   struct A{
3      A(int a){std::cout<<"A(i)"<<std::endl;}
4      A(){std::cout<<"A()"<<std::endl;}
5   };
6   struct B : public A {
7      int a;
8      void methode(){}
9
10  };
11  int main(){
12     A a{3};
13     B b1;
14     B b2{3};
15     return 0;
16  }
```

Réponse : □

En l'état ce code est compilable sans erreur. Si on décommente la ligne commenté alors il n'est plus compilable. Pourquoi ?

```
1   #include<iostream>
2   struct A{
3      A(int a){std::cout<<"A(i)"<<std::endl;}
4      A(){std::cout<<"A()"<<std::endl;}
5   };
6   struct B : public A {
7      int a;
8      void methode(){}
9      B(){ std::cout<<"B()"<< std::endl;}
10  };
11  int main(){
12     A a{3};
13     B b1;
14     //B b2{3};
15     return 0;
16  }
```

Réponse : □

Exercice. *Héritage 1 H1I4*

Ce code produit-il une erreur ? Si oui pourquoi ? Si non qu'affiche-t-il ?

```
1  #include<iostream>
2  struct A{
3      private:
4          int a=1;
5      public :
6          void afficher(){
7              std::cout << a << std::endl;
8          }
9  };
10 struct B : public A {
11     void methode(){ a=3; }
12 };
13 int main(){
14     B b;
15     b.afficher();
16     b.methode();
17     b.afficher();
18     return 0;
19 }
```

Réponse : □

Ce code produit-il une erreur ? Si oui pourquoi ? Si non qu'affiche-t-il ?

```
1  #include<iostream>
2  struct A{
3      protected:
4          int a=1;
5      public :
6          void afficher(){
7              std::cout << a << std::endl;
8          }
9  };
10 struct B : public A {
11     void methode(){ a=3; }
12 };
13 int main(){
14     B b;
15     b.afficher();
16     b.methode();
17     b.afficher();
18     return 0;
19 }
```

Réponse : □

Exercice. *Héritage 1 H1I7*

À quelle ligne et pourquoi le code suivant produit-il une erreur ?

```
1  #include<iostream>
2  struct A{
3      protected:
4          int a;
5  };
6  struct B : public A {
7      void methode()
8      {
9          A a;
10         B b;
11         b.a=2;
12         this->a=0;
13         a.a=3;
14     }
15 };
16 int main(){
17     return 0;
18 }
```

Réponse : □

Référence 3 R3I

Exercice. *Référence 3 R3I1*

Ce code compile sans erreur, qu'affiche son exécution ? Si on décommente la ligne (commentée) le code compile-t-il ?

```
1  #include<iostream>
2  int main(){
3      int i=0;
4      int & ri=i;
5      int && rri=5;
```

```
6     // int& r2i=4;
7     std::cout << rri << std::endl;
8     rri=4;
9     std::cout << rri << std::endl;
10    return 0;
11 }
```

Réponse : □

Exercice. *Référence 3 R3I2*

Ce code compile sans erreur, qu'affiche son exécution ? Si on décommente la ligne (commentée) le code compile-t-il ? Si on commente la fonction f1 le code compile-t-il toujours ?

```
1  #include<iostream>
2  #include<utility>
3  void f(int& a){std::cout<<"f1"<<std::endl;}
4  void f(int&& a){std::cout<<"f2"<<std::endl;}
5  void f(const int&& a){std::cout<<"f3"<<std::endl;}
6  void f(const int& a){std::cout<<"f4"<<std::endl;}
7  int main(){
8     int a;
9     const int b=1;
10    f(3);
11    f(a);
12    //f(b);
13    f(std::move(a));
14    f(std::move(b));
15    return 0;
16 }
```

Réponse : □

Template 1 T1I

Exercice. *Template 1 T1I1*

Ce code produit une erreur de compilation, à quelle ligne et pourquoi ?

```
1  struct A{};
2  template <typename T> void f(T a) { }
3  int main()
4  {
5     A a;
6     f<int>(a);
7     return 0;
8  }
```

Réponse : □

Exercice. *Template 1 T1I4*

Ce code produit-il une erreur de compilation ? Si oui pourquoi ? Si non quel est le type de l'expression `f<double>(...)` ?

```
1  template <typename T> T f(T a) {
2     return a*2;
3  }
4  int main()
5  {
6     f<double>(3);
7     return 0;
8  }
```

Réponse : □

Exercice. *Template 1 T1I2*

Ce code compile sans erreur, qu'affiche son exécution ?

```
1  #include<iostream>
2  struct A{
3      void f(){
4          std::cout << "A" << std::endl;
5      }
6  };
7  struct B{
8      void f(){
9          std::cout << "B" << std::endl;
10     }
11 };
12 template <typename T> void f(T a) {
13     a.f();
14 }
15 int main()
16 {
17     A a;
18     B b;
19     f(a);
20     f(b);
21     return 0;
22 }
```

Réponse : □

Exercice. *Template 1 T1I3*

Ce code produit une erreur de compilation, à quelle ligne et pourquoi ?

```
1  #include<iostream>
2  struct Clb{
3          void m(){ std::cout << "B" << std::endl; }
4  };
5  struct Cla{
6          void n(){ std::cout << "A" << std::endl; }
7  };
8
9  template <typename T> void f(T a)
10 {
11     a.m();
12 }
13
14 int main()
15 {
16     Clb b;
17     Cla a;
18     f(a);
19     f(b);
20     return 0;
21 }
```

Réponse : □

Exercice. *Template 1 T1I5*

Ce code compile sans erreur, qu'affiche son exécution ? à quelle ligne et pourquoi ?

```
1  #include<iostream>
2  struct A{};
3  template <typename Bidule,typename Gachin> Gachin operator || (Gachin a,Bidule b)
```

```
{
    return a;
}
void f(int a){std::cout << "int" << std::endl;}
void f(A a){std::cout << "A" << std::endl;}

int main()
{
    A a;
    int b=1;
    f(b || a);
    f(a || b);
    return 0;
}
```

Réponse : □

Héritage 2 H2I

Exercice. *Héritage 2 H2I1*

Ce code est compilable et s'exécute normalement. Qu'affiche l'exécution ?

```
#include<utility>
#include<iostream>

using namespace std;

struct A{
    virtual void f(){cout << "je suis f1 " << endl;}
            void g(){cout << "je suis g1 " << endl;}
};
struct B : public A {
            void f(){cout << "je suis f2 " << endl;}
            void g(){cout << "je suis g2 " << endl;}
};

int main(){
    A* a1=new B();
    B* b1=new B();
    B b;
    A& a2=b;
    a1->g();
    a1->f();
    b1->f();
    b1->g();
    a2.f();
    a2.g();
}
```

Réponse : □

Héritage Multiple HMI

Exercice. *Héritage Multiple HMI1*

Ce code est compilable et s'exécute normalement. Qu'affiche l'exécution ?

```
#include<utility>
#include<iostream>

```

```
4  using namespace std;
5  struct A {
6     A(){cout << "constr A" << endl;}
7     ~A(){cout << "destr A" << endl;}
8  };
9  struct B {
10    B(){cout << "constr B" << endl;}
11    ~B(){cout << "destr B" << endl;}
12 };
13 struct C  {
14    C(){cout << "constr C" << endl;}
15    ~C(){cout << "destr C" << endl;}
16 };
17 struct D  {
18    D(){cout << "constr D" << endl;}
19    ~D(){cout << "destr D" << endl;}
20 };
21 struct E : public C, public A  {
22    B b;
23    D d;
24 };
25 int main(){ E e; }
```

Déplacement DI

Exercice. *Déplacement DI1*

Ce code compile et s'exécute correctement. Qu'affiche-t-il ?

```
1  #include<utility>
2  #include<iostream>
3  using namespace std;
4  struct A{
5     A(){ cout << "je suis le constructeur par defaut" << endl;}
6     A(int a){ cout << "je suis le constructeur a partir de int" << endl;}
7     A(const A& s){ cout << "je suis le constructeur de copie" << endl; }
8     A(A&& a){ cout << "je suis le constructeur par deplacement" << endl; }
9     A& operator =(const A& s){ cout << "je suis l'operateur d'affectation"
10       << endl; return *this;}
11    A& operator =(A&& s){ cout << "je suis suis l'operateur de deplacement "
12        << endl; return *this;}
13    ~A(){ cout << "je suis le destructeur" << endl; }
14 };
15
16 void f(A a){}
17
18 int main(){
19    A a;
20    A b=5;
21    A c=a;
22    A d=move(a);
23    f(3);
24    f(b);
25    f(move(c));
26    c=b;
27    c=move(a);
28 }
```

Réponse : □