***[C++ FAQ Lite fr](http://jlecomte.ifrance.com/c++/c++-faq-lite/index-fr.html)*,** [**Copyright © 1991-2002**](http://jlecomte.ifrance.com/c++/c++-faq-lite/copy-permissions-fr.html#[1.2])**,** [**Marshall Cline**](http://www.parashift.com/)[**cline@parashift.com**](mailto:cline@parashift.com) **)**

*Traduit de l'anglais par* [*Fabrice Clerc*](mailto:fclerc@cybercable.com)

**1. Un destructeur, c'est quoi au juste?**

Les destructeurs sont utilisés pour libérer les ressources allouées par un objet. Par exemple, une classe Lock pourrait acquérir un sémaphore, et ce sémaphore serait relâché par le destructeur de la classe. Mais l'exemple le plus classique, c'est quand le constructeur fait un new et que le destructeur fait un delete.

Les destructeurs sont des fonctions qui avertissent l'objet qu'il doit se "préparer à mourir". On trouve souvent l'abréviation "dtor" pour signifier destructeur.

**2. Dans quel ordre les objets locaux sont-ils détruits?**

Dans l'ordre inverse de celui dans lequel ils ont été construits: le premier objet construit est le dernier détruit.

Dans l'exemple ci-dessous, le destructeur de b sera exécuté en premier, suivi du destructeur de a:

    void userCode()  
    {  
      Fred a;  
      Fred b;  
       *// ...*  
    }

**3. Dans quel ordre les objets contenus dans un tableau sont-ils détruits?**

Dans l'ordre inverse de celui dans lequel ils ont été construits: le premier objet construit est le dernier détruit.

Dans l'exemple ci-dessous, l'ordre des destructions est a[9], a[8], ..., a[1], a[0]:

    void userCode()  
    {  
      Fred a[10];  
       *// ...*  
    }

**4. Peut-on surcharger un destructeur?**

Non.

Une classe ne peut avoir qu'un seul destructeur. Pour une classe Fred, par exemple, le destructeur est toujours Fred::~Fred(). Un destructeur ne prend pas de paramètres et ne retourne jamais quoi que ce soit.

De toutes les façons, on ne peut pas passer de paramètres à un destructeur puisqu'il n'est jamais appelé explicitement (enfin disons, [*presque* jamais).](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.10])

**5. Doit-on détruire explicitement les objets locaux?**

Surtout pas!

Car le destructeur sera appelé *une deuxième fois* au niveau de l'accolade fermant le bloc dans lequel l'objet a été créé. La norme C++ le garantit et vous ne pouvez rien faire pour empêcher que ça arrive; c'est *automagique*. Et ça risque de vraiment *très mal* se passer si le destructeur d'un objet est appelé deux fois de suite. Pan! Vous êtes mort!

**6. Et si on veut absolument qu'un objet local "meure" avant l'accolade fermant le bloc dans lequel il a été créé? Peut-on appeler explicitement le destructeur si c'est *vraiment* nécessaire?**

Non! (Lire [la réponse précédente](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.5]) pour situer le contexte).

Imaginez que la destruction d'un objet local de type File ait pour effet la fermeture du fichier (c'est souhaitable). Imaginez maintenant que vous ayez un objet f de la classe File, et que vous vouliez que le fichier soit fermé avant la fin du bloc dans lequel se trouve cet objet f (c'est-a-dire avant l'accolade fermant le bloc):

    void someCode(){  
      File f;  
      
       *// ... [Ici, le fichier est encore ouvert] ...*  
      
       *// <— On veut faire ici comme si on détruisait l'objet!*  
      
       *// ... [Ici, le fichier est fermé] ...*  
    }

Ce problème a une [solution simple](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.7]). Souvenez-vous: [*n'appelez jamais explicitement un destructeur!*](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.5])

Il suffit de limiter la durée de vie de l'objet local en le plaçant dans un bloc { ... } artificiel:

    void someCode(){  
      {  
        File f;  
         // ... [Ici, le fichier est encore ouvert] ...  
      }  
     // ^— Ici, le destructeur de *f* est appelé automagiquement!  
      
       // ... [Le code ici s'executera après que f soit fermé]  ...  
    }

**7. OK, OK, j'ai compris: on ne peut pas détruire explicitement un objet local. Mais comment faire alors pour résoudre le problème présenté juste au-dessus?**

(Lire [la réponse précédente](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.6]) pour situer le contexte)

Il suffit de limiter la durée de vie de l'objet local en le plaçant dans un bloc { ... } artificiel:

    void someCode()  
    {  
      {  
        File f;  
         *// ... [Ici, le fichier est encore ouvert] ...*  
      }  
     *// ^— Ici, le destructeur de f est appelé automagiquement!*  
      
       *// ... [Le code ici s'exécutera après que f soit fermé]  ...*  
    }

**8. Et s'il n'est pas possible de placer l'objet local dans un bloc artificiel?**

Dans la plupart des cas, il est possible de limiter la durée de vie d'un objet local en le [plaçant dans une bloc artificiel ({ ... })](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.7]) . Si, pour une raison ou pour une autre, ce n'est pas possible, ajoutez à la classe une fonction membre qui a le même effet que le destructeur. Mais *n'appelez pas le destructeur vous-même!*

Dans le cas de File, par exemple, vous pourriez ajouter à la classe une méthode close(). Le destructeur se contenterait simplement d'appeler cette méthode. Notez que la méthode close() aura besoin de marquer l'objet File de façon à ne pas tenter de fermer le fichier s'il l'est déjà, ce qui peut se produire si close() est appelée plusieurs fois. L'une des solutions possibles est de donner à la donnée membre fileHandle\_ une valeur qui n'a pas de sens, par exemple -1, et de vérifier à l'entrée de la méthode que fileHandle\_ n'est pas égale à cette valeur:

  class File {  
    public:  
      void close();  
      ~File();  
       *// ...*  
    private:  
      int fileHandle\_;    *//  fileHandle\_ >= 0 seulement si le fichier est ouvert*  
    };  
    File::~File(){  
      close();  
    }  
      
    void File::close(){  
      if (fileHandle\_ >= 0) {  
         *// ... [Utiliser les appels système qui conviennent pour fermer le fichier] ...*  
        fileHandle\_ = -1;  
      }  
    }

Notez que les autres méthodes de la classe File peuvent elles aussi avoir besoin de vérifier que fileHandle\_ n'est pas égale à -1 (c'est-à-dire, de vérifier que le fichier n'est pas fermé).

**9. Peut-on détruire explicitement un objet alloué par new?**

Pas dans la plupart des cas.

À moins que vous ayez utilisé [placement new](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.10]) , utilisez delete plutôt que d'appeler explicitement le destructeur de l'objet. Imaginez par exemple que vous ayez alloué un objet grâce à une "new expression" classique:

    Fred\* p = new Fred();

Le destructeur Fred::~Fred() va être appelé automagiquement quand vous utiliserez delete:

    delete p; // p->~Fred()*est appelé automagiquement*

*N'appelez pas* explicitement le destructeur, car cela ne libèrera pas la mémoire allouée pour l'objet Fred lui-même. Gardez à l'esprit que [delete p a deux effets](http://jlecomte.ifrance.com/c++/c++-faq-lite/freestore-mgmt-fr.html#[16.8]) : il appelle le destructeur et il désalloue la mémoire.

**10. Qu'est-ce que "placement new" et dans quels cas l'utilise-t-on?**

On peut utiliser placement new dans de nombreux cas. L'utilisation la plus simple permet de placer un objet a une adresse mémoire précise. Pour cela, l'adresse choisie est représentée par un pointeur que l'on passe à la partie new de la new expression:

    #include <new.h>       *// On doit inclure <new.h> pour utiliser "placement new"*  
    #include "Fred.h"      *// Déclaration de la classe  Fred*  
      
    void someCode()  
    {  
      char memory[sizeof(Fred)];      *// Ligne 1*  
      void\* place = memory;           *// Ligne 2*  
      
      Fred\* f = new(place) Fred();    *// Ligne 3 (voir "DANGER" ci-dessous)*  
       *// Les deux pointeurs f et  place sont maintenant égaux*  
      
       *// ...*  
    }

La ligne 1 crée un tableau dont la taille en octets est sizeof(Fred), tableau donc assez grand pour que l'on puisse y stocker un objet de type Fred. La ligne 2 crée un pointeur place qui pointe sur le premier octet de cette zone mémoire (les programmeurs C expérimentés auront noté que cette deuxième étape n'était pas strictement nécessaire; en fait, elle est là juste pour rendre le code plus lisible). Pour faire simple, on peut de dire de la ligne 3 qu'elle appelle le constructeur Fred::Fred(). Dans ce constructeur, this et place ont la même valeur. Le pointeur f retourné sera donc lui aussi égal à place.

**CONSEIL:** n'utilisez pas cette syntaxe du "placement new" si vous n'en avez pas l'utilité. Utilisez-là uniquement si vous avez besoin de placer un objet à une adresse mémoire précise. Utilisez-là par exemple si le matériel sur lequel vous travaillez dispose d'un périphérique de gestion du temps mappé en mémoire à une adresse précise, et que vous voulez placer un objet Clock à cette adresse.

**DANGER:** il est de votre *entière* responsabilité de garantir que le pointeur que vous passez à l'opérateur "placement new" pointe sur une zone mémoire assez grande et correctement alignée pour l'objet que vous voulez y placer. Ni le compilateur ni le run-time de votre système ne vérifient que c'est effectivement le cas. Vous pouvez vous retrouver dans une situation fâcheuse si votre classe Fred nécessite un alignement sur une frontière de 4 octets et que vous avez utilisé une zone mémoire qui n'est pas correctement alignée (si vous ne savez pas ce qu'est "l'alignement", alors n'utilisez pas la syntaxe du "placement new").

La destruction de l'objet ainsi créé est aussi sous votre entière responsabilité. Pour détruire l'objet, il faut appeler explicitement son destructeur:

    void someCode()  
    {  
      char memory[sizeof(Fred)];  
      void\* p = memory;  
      Fred\* f = new(p) Fred();  
       *// ...*  
      f->~Fred();    *// Appel explicite au destructeur*  
    }

C'est un des très rares cas d'appel explicite au destructeur.

**11. Dans le code d'un destructeur, doit-on détruire explicitement les objets membres?**

Non. Il n'est jamais nécessaire d'appeler explicitement un destructeur (sauf si l'objet a été créé avec un [placement new](http://jlecomte.ifrance.com/c++/c++-faq-lite/dtors-fr.html?2&weborama=-1#[11.10])).

Le destructeur d'une classe (il existe même si vous ne l'avez pas défini) appelle *automagiquement* les destructeurs des objets membres. Ces objets sont détruits dans l'ordre inverse de celui dans lequel ils apparaissent dans la déclaration de la classe.

    class Member {  
    public:  
      ~Member();  
       *// ...*  
    };  
      
    class Fred {  
    public:  
      ~Fred();  
       *// ...*  
    private:  
      Member x\_;  
      Member y\_;  
      Member z\_;  
    };  
      
    Fred::~Fred()  
    {  
       *// Le compilateur appelle automagiquement  z\_.~Member()*  
       *// Le compilateur appelle automagiquement  y\_.~Member()*  
       *// Le compilateur appelle automagiquement  x\_.~Member()*