



Centre Informatique pour les **L**ettres
et les **S**ciences **H**umaines

Apprendre C++ avec Qt : Annexe 4

Surcharger les opérateurs new et delete

1 - L'opérateur new

Il est possible de surcharger l'opérateur new soit globalement soit spécifiquement pour une classe. En cas de surcharge globale, la fonction opérateur sera appelée lors de chaque utilisation de new, et doit donc être assez générale pour allouer dynamiquement la mémoire nécessaire¹, quel que soit le type de l'objet dont la création est demandée. Lorsque l'opérateur new est surchargé par une fonction membre, en revanche, celle-ci ne sera appelée que lors de la création dynamique d'une instance de la classe en question.

Caractéristiques d'une fonction operator new()

Qu'elle soit globale ou membre d'une classe, une fonction operator new() doit respecter certaines contraintes liées à la nature un peu particulière de l'opération qu'elle effectue.

- Toute fonction surchargeant new doit être de type "pointeur générique" et doit renvoyer l'adresse d'une zone de mémoire allouée dynamiquement par ses soins.

Le type "pointeur générique" ne sera étudié en détail que dans la Leçon 22. En l'occurrence, son utilisation comme type de la fonction operator new() n'a guère de conséquences, car la valeur renvoyée est automatiquement transtypée lors de son affectation au pointeur auquel elle est destinée (il s'agit là d'un cas exceptionnel : l'affectation d'une valeur de type "pointeur générique" à une variable d'un autre type exige normalement un transtypage explicite).

- Une fonction surchargeant new peut recevoir un ou plusieurs paramètres.
- Le premier paramètre de toute fonction surchargeant new doit être de type `size_t`², et la valeur servant à initialiser ce paramètre ne doit pas être fournie lors de l'appel de la fonction (elle sera automatiquement générée par le compilateur : il s'agit en fait de la taille de la zone de mémoire requise).
- Etant donné que la fonction operator new() ne saurait être invoquée au titre de l'objet concerné (qui n'existe pas encore...), cette fonction est automatiquement rendue statique (cf. Annexe 3) si elle est membre d'une classe.

Le processus de création dynamique d'une instance

Il est important de bien comprendre que l'exécution d'une fonction operator new() n'est que la première étape de la création dynamique d'une instance. Une fois cette exécution terminée, un constructeur est automatiquement appelé, de façon à initialiser la zone de mémoire qui vient d'être réservée. La version du constructeur qui sera utilisée dépend des arguments éventuellement fournis. Ainsi, si

```
CTruc *unTruc = new CTruc;
```

appelle la fonction operator new(), puis le constructeur par défaut,

```
CTruc *unTruc = new CTruc(17);
```

appellera la fonction operator new(), puis le constructeur de transtypage d'int vers CTruc (ce qui suppose, bien entendu, qu'un tel constructeur ait été défini).

¹ Il ne serait VRAIMENT pas raisonnable d'envisager une fonction surchargeant new et n'assurant pas l'équivalent fonctionnel d'une allocation dynamique de mémoire en vue de la création d'un objet...

² La définition du type `size_t` est donnée dans le fichier `stddef.h`. Il s'agit d'un simple alias destiné à masquer au code source le fait que, selon les systèmes, la taille des objets s'exprime sur deux ou quatre octets.

Exemples

Voici une fonction `operator new()` globale, qui pourrait servir à étudier le comportement (ou même à déboguer...) un programme utilisant l'allocation dynamique.

```
1 void * operator new(size_t taille)
2 {
3     void *adresse = malloc(taille);
4     cout << "quelqu'un vient d'allouer " << taille << " octets\n";
5     return adresse;
6 }
```

La fonction ci-dessus obtient l'allocation d'une zone de mémoire de la taille nécessaire en faisant appel à la fonction `malloc()`. Cette fonction fait partie de la librairie standard du langage C et, à ce titre, est disponible pour un programme C++, pour peu qu'il y figure une directive `#include "stdlib.h"`. Dans un programme réel, on peut penser que, si le programmeur se donne la peine de surcharger `new`, c'est parce qu'il a une idée bien précise de la façon dont il souhaite que la gestion de la mémoire soit effectuée. Il sera bien entendu alors conduit à mettre en place un dispositif plus complexe qu'un simple appel à `malloc()`.

Si on s'intéresse plus particulièrement à la classe `CTruc`, on peut se contenter d'examiner les demandes d'allocation dynamique concernant les objets de ce type :

```
1 void * CTruc::operator new(size_t taille)
2 {
3     void *adresse = malloc(taille);
4     cout << "quelqu'un vient de réclamer l'allocation d'un CTruc\n";
5     return adresse;
6 }
```

Utilisation d'un opérateur new surchargé

Qu'il s'agisse d'une fonction globale ou d'un membre d'une classe, l'usage d'une fonction `operator new()` ne fait pas intervenir une syntaxe différente de celle devant être utilisée lors d'une allocation dynamique utilisant l'opérateur `new` prédéfini :

```
CTruc * ptr = new CTruc;
```

Lorsque la fonction `operator new()` comporte plusieurs arguments, il est bien entendu indispensable de transmettre, lors de l'appel de la fonction, les valeurs destinées à l'initialisation de ces paramètres. Si l'on suppose que la classe `CTruc` comporte une fonction `operator new()` définie ainsi :

```
1 void * CTruc::operator new(size_t taille, double val)
2 {
3     //le code inséré ici peut utiliser val comme il l'entend
4 }
```

On transmettra la valeur requise en écrivant

```
CTruc * ptr = new(5.2) CTruc;
```

Il ne faut pas confondre les valeurs destinées aux **paramètres de l'opérateur new** avec les **valeurs qui doivent être transmises au constructeur** qui sera appelé une fois l'allocation de mémoire effectuée. Si la classe `CTruc` comporte, en plus de l'opérateur `new` décrit ci-dessus, un constructeur de transtypage à partir d'un double, on pourra être amené à écrire :

```
CTruc * ptr = new(5.2) CTruc(3.14);
```

2 - L'opérateur delete

Une fonction `operator delete()` ne doit pas renvoyer de valeur. Elle doit accepter un paramètre de type "pointeur générique" qui indique l'adresse de la zone de mémoire à libérer et, éventuellement, un paramètre de type `size_t` qui donnera la taille de cette zone. Ces deux valeurs sont automatiquement fournies par le compilateur et n'ont donc pas à figurer dans le

code appelant. L'exécution d'une fonction `operator delete()` est la dernière étape de la destruction d'une instance créée de façon dynamique : elle est précédée de l'appel du destructeur de la classe concernée. Si `operator new()` est défini ainsi :

```
1 void * CTruc::operator new(size_t taille)
2 {
3     void *adresse = malloc(taille);
4     cout << "quelqu'un vient de réclamer l'allocation d'un CTruc" << endl;
5     return adresse;
6 }
```

il semble logique que `operator delete()` le soit ainsi :

```
1 void CTruc::operator delete(void * aTuer, size_t taille)
2 {
3     free(aTuer); //free() est la fonction libérant la mémoire allouée par malloc()
4     cout << "mort du CTruc qui habitait à l'adresse " << aTuer << endl;
5     cout << taille << " octets ont été libérés." << endl;
6 }
```