**Compiler une application Qt**

Le mieux, c'est d'utiliser QtCreator qui est tout bien configuré pour cela.

Sinon, vous devez utiliser la ligne de commande configurée pour Qt qui vient avec l'installation du SDK. Dans cette ligne de commande, toutes les variables d'environnement sont bien déclarées.

Allez chercher la ligne de commande Qt dans :

* Démarrer | Qt by Nokia v4.x.x Command Prompt

Déplacez-vous dans le répertoire de votre projet

* **cd Répertoire/de/votre/projet/** (\ backslash pour Windows)

Exécutez la commande qmake -project pour générer le fichier projet.pro :

* **qmake -project**

Exécutez la commande qmake nomProjet.pro ou qmake -tp vc selon l'OS :

* **qmake nomProjet.pro** (Linux)
* **qmake -tp vc** (Wnidows pour créer un projet visual studio)

Compilez et exécutez le projet :

* **make** (Linux)
* Ouvrez le projet dans Visual Studio et exécutez le (Windows)

Note pour Visual Studio : Si à l'exécution du projet, Visual Studio vous dit qu'il ne trouve pas l'exécutable, allez dans les propriétés du projet et changez l'emplacement de l'exécutable :

* Projet | Propriétés
* Propriétés de configuration | Général
* Répertoire de sortie -> écrivez "debug\"

## Exécution d'un programme sans system("pause")

### Sous Visual Studio 2008 et les versions antérieures :

Pas besoins de system("PAUSE"), premièrement c'est inutile et surtout ce n'est pas compatible avec d'autres systèmes d'exploitation que Windows.

Vous avez tout simplement besoins d'exécuter le programme en utilisant le bouton 'Exécuter sans débogage' (ou le raccourci Ctrl-F5) dont l'icône est une flêche vide. Elle se trouve généralement dans l'onglet Déboguer.

### Sous Visual Studio 2010 :

Pas besoins de system("PAUSE"), premièrement c'est inutile et surtout ce n'est pas compatible avec d'autres systèmes d'exploitation que Windows.

Vous devez faire cette procédure pour chaque projet :

1. Sélectionnez le projet dans l'Explorateur de solutions
2. Cliquez sur l'onglet « Projet -> Propriétés »
3. Dans « Propriétés de configuration -> Éditeur de liens -> Système »
4. Dans le menu déroulant de Sous-système (première propriété)
5. Choisissez Console (/SUBSYSTEM:CONSOLE)

## Copie d'une variable vers un pointeur

On doit faire attention lorsqu'on copie le contenu d'une variable vers un pointeur. On a deux choix dépendamment de ce que nous avons besoins :

// Lorsqu'on veut seulement avoir accès à la valeur de la variable  
// On fait une copie de l'adresse  
pInt = &monInt;

// Lorsqu'on veut faire une copie profonde de la variable  
if(pInt == 0)  
pInt = new int;  
\*pInt = monInt;

## Utiliser 0 ou NULL

La réponse de Bjarne Stroustrup à la question "Should I use NULL or 0?":

In C++, the definition of NULL is 0, so there is only an aesthetic difference. I prefer to avoid macros, so I use 0. Another problem with NULL is that people sometimes mistakenly believe that it is different from 0 and/or not an integer. In pre-standard code, NULL was/is sometimes defined to something unsuitable and therefore had/has to be avoided. That's less common these days.  
If you have to name the null pointer, call it nullptr; that's what it's going to be called in C++0x. Then, "nullptr" will be a keyword.

Sources : <http://www2.research.att.com/~bs/bs_faq2.html#null>

## Quand est-ce qu'un objet doit appeler delete

Il y a plusieurs cas possibles. Généralement, on appelle l'opérateur delete au même niveau que nous avons appelé l'opérateur new. Donc, si nous avons créé dynamiquement un objet dans le main(), on doit le détruire dans le main(). Si un objet alloue de la mémoire pour un tableau ou un autre objet dans son constructeur, il doit désollouer la mémoire dans son destructeur. Mais si on on alloue de la mémoire dans l'appel du constructeur, c'est l'objet construit qui doit désallouer la mémoire :

MaClasse monObjet = Maclasse( new int(5) ); // OU

MaClasse\* monObjet = new Maclasse( new int(5) );

Une autre règle très utile est de considérer s'il s'agit de composition ou d'aggrégation. Si c'est de la composition, l'objet doit toujours s'occuper de désallouer la mémoire. Sinon, il ne doit rien faire.

## Destruction d'un objet contenant un tableau

Tout d'abord, il faut dissocier l'objet de son contenu. Un objet est toujours créé de la même façon peu importe son contenu. Si ses attributs nécessitent une manipulation particulière, c'est au constructeur et au destructeur de s'en occuper.

class TableauDynamique {  
public :

TableauDynamique();  
~TableauDynamique();

private :

int\* tableau\_;

};  
  
TableauDynamique::TableauDynamique() {

tableau\_ = new int[5];

}  
  
TableauDynamique::~TableauDynamique() {

delete [] tableau\_;

}

#### Pour construire et détruire un objet de la classe TableauDynamique :

TableauDynamique \*table = new TableauDynamique();  
delete table;

#### À éviter pour la destruction d'un objet de la classe TableauDynamique :

TableauDynamique \*table = new TableauDynamique();  
delete [] table;

## Implémentation dans le fichier .h

Il n'est pas interdit d'implémenter quelques unes ou toutes les méthodes dans le fichier de définition de la classe. Ceci est toutefois déconseillé. Il est préférable de séparer la définition de l'implémentation, car cela rend la lecture d'une classe plus claire et plus rapide. Malheureusement, la conception de classes génériques nous oublige parfois à placer l'implémentation dans le .h, mais on placera quand même celle-ci en dehors de la définition.

## Dans une boucle : iter<cont.end() ou iter!=cont.end()

On pourrait croire qu'il est préférable d'utiliser iter<cont.end() pour être sûr que l'on a pas dépassé la fin du conteneur, mais ce ne sont pas tous les types d'itérateur qui supportent le concept de distance relative. En effet, seul les itérateurs à accès aléatoire le supportent. On doit alors toujours utiliser l'expression iter!=cont.end() dans les boucles while et for.

## Construction d'objet dans un bloc try

Si nous savons que le constructeur peur lever une exception, il faut mettre la construction de cet objet dans un bloc try avec le gestionnaire approprié. Dans ce cas ci, notre objet n'est pas accèssible en dehors du bloc try. Nous ne pourons donc pas appeler les méthodes de cet objet en dehors du bloc. Ce n'est pas grave, puisque si la construction échoue, pourquoi voudrions-nous l'utiliser. Donc, tous les appels de méthodes sur un objet construit dans un bloc try, doivent être dans le même bloc try.

## erreur C2259: 'MaClasse' : impossible d'instancier une classe abstraite

Vous savez qu'il n'est pas possible d'instancier une classe abstraite. Une classe est dite abstraite lorsque celle-ci possède une méthode virtuelle pure. Mais si votre classe n'en possède pas et que vous recevez quand même cette erreur, c'est probablement parce que celle-ci hérite d'une classe abstraite et que vous avez oublié d'implémenter la méthode virtuelle pure héritée.

## error C3892: you cannot assign to a variable that is const

Il est souvent très facile de corriger cette erreur, mais parfois, aucun indice nous laisse croire qu'une de nos opérations manipule des objets const. Un des cas les plus difficile à repérer est la manipulation de set (ou d'un map) de la STL. Il faut se rappeler que les éléments d'un set sont imuables puisque modifier un élément comporte souvent de changer sa position dans le conteneur. De ce fait, on ne peut pas appeler l'algorithme set\_union comme cela :

// L'iterateur monSet.begin() est en fait un const\_iterator  
set\_union(monCont1.begin(), monCont1.end(), monCont2.begin(), monCont2.end(), monSet.begin());

Ce que vous devez faire est de donner un introducteur vers le set comme dernier argument :

// inserter(monSet, monSet.begin()) est l'introducteur  
set\_union(monCont1.begin(), monCont1.end(), monCont2.begin(), monCont2.end(), inserter(monSet, monSet.begin()));

## Symboles externes non résolus

Généralement, cette erreur soulevée par le compilateur indique que des méthodes déclarées dans la définition de la classe ne possèdent pas d'implémentation. Cela peut aussi être dû à un attribut statique qui n'a pas été initialisée dans le fichier .cpp.