

Precompiled headers

Christophe [Groove] Riccio

www.g-truc.net

Création : 24/02/2006

Introduction

Ce document a pour but de présenter l'utilisation des précompiled headers lors d'une utilisation avec Visual Studio et GCC. Ce mécanisme permet de diminuer le temps de compilation d'un programme de manière significative.

1. Le principe des precompiled headers

La première phase de la compilation d'un fichier source consiste à générer un header contenant tous les informations de dépendance nécessaire à ce fichier source. Nous pouvons légitimement considérer que ce header est ensuite placé au début du fichier source lors d'un processus de compilation usuel.

Que contient ce header ? Dans le cas d'une classe il est d'usage de séparer sa déclaration et sa définition qui sont respectivement placés dans le fichier d'en-tête et dans le fichier source. L'utilisation de l'instruction `#include` place naturellement la déclaration de la classe au début du fichier source lors de la compilation. De même toutes les dépendances utilisant `#include` ce retrouveront dans ce header ce qui inclut la bibliothèque STL mais aussi toutes les bibliothèques externes. En conséquence, le fichier source peut devenir énorme et chaque fichier source dispose de son header personnalisé. A vrai dire une grande partie du temps de compilation est dédié à traiter ce processus de création d'un header.

Ce qui fait souffrir, c'est que ce processus est répété pour chaque fichier source. Une première utilisation des précompiled header consiste à créer un précompiled header pour chaque fichier source. Si le fichier d'en-tête du fichier source n'est pas modifié et qu'aucun « include » n'est ajouté alors le header n'est pas recréé. Une autre utilisation des précompiled headers repose sur l'idée de se limiter à la création d'un seul header pour tous les fichiers sources.

2. Utilisation d'un precompiled header par fichier source

La méthode reposant sur le concept d'un precompiled header par source n'a aucune implication sur le code, l'organisation du projet ou l'optimisation mais son intérêt reste limité à un cas bien précis : la modification seule du contenu du fichier source. Sous GCC, il suffit de compiler un fichier source, il sera alors automatiquement utilisé lors de la compilation du fichier source. (`file.h`, `file.h.pch`, `file.cpp`). Sous Visual Studio, il faut activer pour tout le projet l'option « automatically generate » dans le menu C/C++ des propriétés du projet.

Cette mode ne me semble pas vraiment d'un grand intérêt. En effet, elle trouve son utilité lorsque le projet a déjà été compilé et que l'on édite seulement le code source. Dans la pratique, c'est assez rare et surtout valable sur un petit nombre de fichiers, le temps de compilations n'est alors pas vraiment important à la base, la perception de l'amélioration paraît null.

3. Utilisation d'un precompiled header seul

Pour parvenir à la création d'un seul header, il est nécessaire d'utiliser un fichier source qui a un rôle de stockage du header. De plus, un fichier, habituellement nommé `precompiled.h`, `pheader.h`, `stdafx.h` (sous VC) ou `all.h` (sous GCC), est chargé de collecter toutes les dépendances. C'est dans ce fichier que sont inclus les bibliothèques externes et tout ce dont nous voulons inclure dans le precompiled header.

Ce header doit alors être inclus par tous les fichiers sources du projet. Si ce point n'est pas appliqué, le compilateur retournera simplement une erreur.

4. Comment faut t'il procéder sous Visual Studio ?

Dans les propriétés du projet sélectionnez l'entrée « precompiled headers » dans la partie C/C++. Choisissez alors l'option d'utilisation d'un fichier pour le précompilé header et indiquez le chemin de votre fichier avec son nom, par exemple « ./precompiled.h ». Vous devez impérativement indiquer le chemin tel que vous l'indiquez dans les fichiers sources. Si votre hiérarchie est complexe, rappelez vous que les compilateurs procèdent des options pour préciser des répertoires pour les fichiers d'en-tête, vous pourrez alors inclure votre fichier ainsi : « #include <precompiled.h> ».

En suite, choisissez un fichier source particulier. Ce fichier source sera utilisé pour créer le précompilé header, vous pouvez le dédier à cet usage, « precompiled.cpp » faisant alors tout à fait l'affaire. Dans les propriétés du projet, sélectionnez alors l'option de création du précompilé header.

/!\ Attention /!\

Il est important de faire attention de sélectionner l'option d'utilisation du précompilé header pour le projet et non un fichier spécifique. Par contre, pour le fichier source dédié à la création du précompilé header, seul ce fichier doit avoir l'option de création du précompilé header. Souvenez vous que l'idée est de créer qu'un seul header pour tous les fichiers.

Cette description est valable pour Visual C++ 6.0, 7.0, 7.1 et 8.0, les fondements n'ayant pas vraiment changé au cours des versions.

5. Comment faut t'il procéder sous GCC ?

Sous GCC, les précompilé headers ne sont disponibles que depuis la version 3.4. Sous GCC le fonctionnement se base sur une compilation du fichier « precompiled.h » pour obtenir un fichier nommé « precompiled.h.gch ». Tout comme sous Visual Studio vous pouvez choisir le nom de ce fichier. Le fichier *.gch doit être placé dans le répertoire du fichier *.h du même nom. Fondamentalement, GCC vérifie si le précompilé header est présent et correct et l'utilise à la place du fichier d'en-tête habituel.

Pour vérifier le bon fonctionnement du procédé GCC propose l'option -Wno-pch, qui affiche des warnings en cas de soucis sur les précompilé headers.

Voici un exemple de compilation du précompilé header, oui on compile bien un fichier d'en tête !

```
precompiled.h.gch: precompiled.h
gcc -Wno-pch -c precompiled.h -o precompiled.h.gch
```

Bien entendu, il est impératif de créer le précompilé header avant l'objet pour qu'il soit pris en compte.

Avec GCC, un problème de gestion des versions apparaît lors que l'on utilise les précompilé headers. En effet, le makefile pour une compilation avec précompilé header ne fonctionne que pour GCC 3.4 et supérieur. Avec GCC 3.3 et inférieur, une erreur apparaît vous signalant la tentative de compiler un fichier d'en-tête.

6. Problématiques liées à l'utilisation des précompilé headers

L'utilisation de précompilé headers peut augmenter significativement la vitesse de compilation d'un programme cependant tout avantage trouve ses inconvénients. En effet les inclusions doivent alors avoir une structure telle que tout fichier connaît l'ensemble des déclarations du projet. Ceci peut poser problème par exemple dans le cas très rare des liaisons circulaires. Cependant, plusieurs approches sont possibles, nous pouvons aussi n'inclure que les bibliothèques externes dans le précompilé header et utiliser une structure habituelle pour le code source du projet, les performances restent très

intéressantes. De plus, l'optimisation de Visual C++ nommée « Whole Program Optimization » n'est alors plus utilisable. Pour GCC, une option similaire ne sera disponible qu'avec la version 4.1 et je ne sais pas ce qu'il en sera.

7. Résultat

Si vous souhaitez découvrir un bon gros exemple d'utilisation des précompiled headers, je vous invite à compiler le SDK de Doom 3 ou de Quake 4 qui les utilise. D'après mes tests sur mon Athlon XP 2800+ avec seulement l'idlib du SDK de Quake 4, la compilation prend 1 minute 25 secondes sans précompiled headers et 10 secondes avec. Plutôt convainquant non ?

Enfin, je mets à votre disposition un exemple simple pour Visual C++ 6.0, 7.1 et 8.0 et MinGW 3.4 disponible à l'adresse www.g-truc.net/article/precompiled.zip.