Les incompatibilités entre le C et le C++

par Nicolas Joseph

Date de publication : 24 Juillet 2005

Dernière mise à jour :

Le C et le C++ sont deux langages proches au niveau de leur syntaxe et de leur grammaire (si l'on met de côté les propriétés objet du C++), cependant il existe un certain nombre d'incompatibilités qui fait qu'un programme écrit en C peut ne pas compiler avec un compilateur C++. La plupart des incompatibilités listées ci-dessous, en particulier celles valables pour le C90, peuvent être évitées avec un compilateur bien configuré et de bonnes habitudes de programmation.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Entre le C (90 et 99) et le C++

Les pointeurs de type void

Un pointeur de type **void** est un pointeur générique. En C, il est possible de réaliser une conversion implicite d'un type donné vers le type **void** et inversement :

```
void *p_void;
int *p_int;
...
p_void = p_int;
p_int = p_void;
```

Compiler ce code avec un compilateur C++, provoque l'erreur suivante :

```
Invalid conversion from `void*' to `int*'
```

Par conséquent, la seconde conversion (d'un pointeur générique vers un pointeur d'un type donné) n'est pas possible implicitement en C++, il faut le faire de manière explicite grâce à l'opérateur de cast :

```
void *p_void;
int *p_int;
...
p_void = p_int;
p_int = (int *)p_void;
```

L'opérateur de cast utilisé dans cet exemple est un opérateur du C autorisé en C++ bien qu'obsolète. Pour un code C++ pur, on préférera le code suivant :

```
void *p_void;
int *p_int;
...
p_void = p_int;
p_int = static_cast<int *>(p_void);
```

L'intruction goto

L'instruction **goto** permet de sauter à une étiquette donnée. Pour exemple, voici un code qui compile parfaitement en C :

```
   int i = 2;
goto start;
   int v = 5;
   printf ("%d\n", v);
start:
   printf ("%d\n", i);
```

Par contre si vous essayez de compiler ceci à l'aide d'un compilateur C++, vous obtiendrez l'erreur suivante :

```
Crosses initialization of `int v'
```

Car en C++, l'instruction **goto** ne peut pas être utilisée pour sauter une déclaration comportant une initialisation sauf si le bloc qui contient cette déclaration est entièrement sauté :

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

```
   int i = 2;
goto start;
   if (i == 2)
   {
      int v = 5;
      printf ("%d\n", v);
   }
start:
   printf ("%d\n", i);
```

Le type d'un caractère

Les types utilisés pour représenter les caractères ne sont pas les mêmes en C et en C++. Pour le premier, il s'agit d'un entier (**int**) alors que pour le second ce sont des **char**. Cela a son importance au niveau de la surchage des fonctions (mécanisme spécifique au C++) puisque dans cet exemple :

```
void printchar (int c)
{
   printf ("%c\n", c);
}

void printchar (char c)
{
   printf ("%c\n", c);
}
```

L'instruction :

```
printchar ('a');
```

Appelera la seconde fonction en C++.

Initialisation de tableaux de caractères

En C, il est possible d'initialiser un tableau de caractères avec une chaîne de caractères de même longueur (sans compter le caractère de fin de chaine \0) :

```
char tab[10] = "Developpez";
```

> *Attention, dans ce cas, on obtient un tableau de caractères et non une chaîne de caractères puisque le caractère \0 n'est pas présent.*

Cette même instruction donnera en C++ :

```
Initializer-string for array of chars is too long
```

Type de retour d'une fonction

En C, si vous omettez le type de retour d'une fonction lors de sa déclaration, le compilateur considère que la fonction retourne un **int** mais en C++ cela n'est pas permis :

```
ISO C++ forbids declaration of `function' with no type
```

> *Un compilateur C bien paramètré vous signalera cet oubli par un warning.*

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Les booléens

Le type booléen est présent en C99 et en C++ cependant en C, si vous n'incluez pas le fichier *stdbool.h* il est tout à fait possible de redéfinir les termes **bool**, **false** et **true**. Par contre en C++ ceci n'est pas permis :

```
Redeclaration of C++ built-in type `bool'
```

Portée de l'opérateur const

En C, la classe d'une variable déclarée constante est, par défaut, externe (**extern**) alors qu'en C++, elle est interne (**static**).

En C :
```
const int i = 1;
/* signifie :
extern const int i = 1; */
```

En C++ :
```
const int i = 1;
/* signifie :
static const int i = 1; */
```

Liste de paramètres vide

Si, lors de la déclaration d'une fonction, la liste des paramètres n'est pas mentionnée, un compilateur C n'effectura pas de vérification. Ainsi le code suivant est correct :

```
void foo() {}

int main (void)
{
  foo ();
  foo (0);
  foo (1, 2);

  return (0);
}
```

Par contre en C++ une liste de paramètres vide est considérée comme valant **void**.

Structure et typedef portant le même nom

En C, la présence du mot **struct**, **enum** ou **union** permet de différencier un **typedef** des types plus complexes (structure, énumération ou union) qui portent le même nom :

```
typedef int var;
struct var { ... }; /* enum ou union revient au même */

int main (void)
{
  var var1;
  struct var var2;
  ...
  return (0);
}
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Ceci ne pose aucun problème en C, par contre en C++, il y a redéfinition.

Les énumérations

En C, une constante énumérée est en fait un entier signé (**signed int**) par conséquent ceci ne pose pas de problème :

```
enum etat { TRAVAIL, MANGE, DORT };

int v1 = TRAVAIL;
enum etat v2 = 1;
```

En C++, la conversion implicite entre une énumération et un entier n'est pas possible, il faut utiliser l'opérateur de cast.

strchr

En C, la fonction **strchr** a le prototype suivant :

```
char* strchr (const char* cs, int c);
```

Le C++ propose deux prototypes pour la fonction **strchr** dans cstring :

```
char *strchr (char *cs, int c);
const char *strchr (const char *cs, int c);
```

Donc le code qui suit est valide en C avec l'en-tête string.h mais invalide en C++ si cstring est incluse :

```
const char *cs = "Developpez";
char *p;
p = strchr (cs, 'v');
```

En effet, puisque *cs* est constant, c'est la première fonction **strchr** qui est appelée, or elle retourne une chaîne constante contrairement à son homologue en C donc pour que le code soit valide, il faut que *p* soit déclaré comme étant constant.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Entre le C90 et le C++

La déclaration implicite de fonction

En C, si une fonction n'est pas explicitement déclarée avant sa première utilisation, le compilateur considère qu'il s'agit d'une fonction externe (**extern**) retournant un entier (**int**) alors que la déclaration est obligatoire en C99 comme en C++.

Déclaration implicite du type d'une variable

En C, il est possible d'omettre le type de la variable lors de son initialisation. Par défaut la variable est considérée comme étant un **int**. Un compilateur C++ vous répondra poliment :

```
ISO C++ forbids declaration of `i' with no type
```

Déclaration de fonction style K&R

Dans des sources ayant un certain âge, on peut trouver ce genre de déclaration :

```
int main (argc, argv)
          int argc; char **argv;
{
   return (0);
}
```

C'est une écriture décrite par Kernighan et Ritchie (d'où l'abréviation K&R) tolérée uniquement en C90.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Entre le C99 et le C++

Passage de tableau comme paramètre d'une fonction

En C, il est possible de déclarer un pointeur constant sur un **int** de cette façon :

```
int *const str;
```

Le C99 apporte une nouvelle manière de le faire (uniquement dans une liste de paramètres d'une fonction) incompatible avec le C++ :

```
void foo (int str[const]);
```

Sera accompagné des erreurs suivantes :

```
Expected primary-expression before "const"
Expected `]' before "const"
Expected `,' or `...' before "const"
```

Initialisation des membres d'une structure

Le C99, permet d'omettre le nom de la variable lors de l'initialisation des membres d'une structure :

```
struct client
{
  char nom[21];
  int cp;
  char pays[21];
};
...
struct client client1 =
{
  .nom = "Paul",
  .cp = 75,
  .pays = "France"
};
```

Ce qui en C++ n'est pas autorisé :

```
Expected primary-expression before '.' token
```

Il faut utiliser le code suivant :

```
struct client client1 =
{
  "Paul",
  75,
  "France"
};
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/

Remerciements

Merci à Anomaly et à Loulou24 pour leurs conseils sur le contenu de cet article

Merci à 2Eurocents et à Beuss pour leurs contributions respectives à la relecture de cet article

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Nicolas Joseph. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://nicolasj.developpez.com/articles/incompatibilites/