Les nouveautés du C99

par Nicolas Joseph

Date de publication : 27 Avril 2006

Dernière mise à jour :

Au travers de cet article, je vous propose un tour d'horizon des nouveautés du C99.

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/c99/

I - Introduction

A l'heure actuelle, deux normes cohabitent : il y a celle publiée en 1989 par l'ANSI**A**merican **N**ational **S**tandard **I**nstitut, reprise en 1990 par l'ISO**I**nternational **O**rganization for **S**tandardization connue respectivement sous le nom de C89 et C90 (ce sont les mêmes) et une seconde norme, qui regroupe les différents *drafts* apparues depuis la dernière norme, publiée en 1999 par ISO connue sous le nom de C99 (et C2k pour la version ANSI). Cette dernière a pour but de minimiser les incompatibilités entre le C et le C++ (sans pour autant en faire un seul et même langage). Voici donc la liste des nouveautés et des modifications apportées par la norme C99 par rapport au 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/c99/

II - Préprocesseur

II-A - La macro __STDC_VERSION__

Cette macro permet de connaître la norme utilisée :

- En C89 elle n'est pas définie
- En C94 elle vaut 199409L
- Et maintenant, en C99, 199901L.

II-B - Macro à nombre variable d'arguments

Cette fonctionnalité permet de définir des macro dont le nombre des arguments peut être différent d'un appel à l'autre. La notation utilisée est la même que pour les fonctions ('...') et la liste est récupérée grâce à la macro *__VA_ARGS__* :

```
#define print_error(s, ...) fprintf (stderr, (s), __VA_ARGS__)
```

II-C - Les commentaires mono-ligne

Les commentaires "à la C++", c'est à dire commençant par // et se terminant à la fin de la ligne viennent s'ajouter aux commentaires classiques (/* */).

II-D - La directive #line

Cette directive du préproceseur permet de modifier le numéro de la ligne courante (et optionnellement le nom du fichier). La valeur utilisée était limitée à 215-1 en C90 et passe maintenant à 231-1.

II-E - La directive #pragma

La directive **#pragma** permet de contrôler le comportement du compilateur. La norme C99 définie trois pragma standards :

- *FP_CONTRACT* : optimisation des expressions flottantes
- *FENV_ACCESS* : notre programme accède aux paramètres du FPU (*Floating Point Unit* ou unité de calcul flottant), ceci se fait à l'aide des fonctions déclarées dans *fenv.h*. Le compilateur doit donc respecter la norme pour les opérations sur les nombres flottant
- *CX_LIMITED_RANGE* : la réduction de l'intervalle n'est pas nécessaire lors de la division de nombres complexes.

Chacune de ces fonctionnalités peuvent être activée (*ON*), désactivée (*OFF*) ou remis à son état par défaut (*DEFAULT*). Par exemple, voici comment activer les optimisations des expressions flottantes :

```
#pragma STDC FP_CONTRACT ON
```

> *Chaque compilateur propose ces propres directives pragma. Si le compilateur rencontre une directive inconnue, la norme prévoie que celle-ci soit ignorée, cependant il est déconseillé d'utiliser des directives non standard car selon les compilateurs leur rôle peut être différent.*

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/c99/

*A la place de la directive **#pragma**, il est possible d'utiliser l'opérateur unaire _**Pragma**.*

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/c99/

III - Syntaxe

III-A - Les nouveaux mots réservés

Le C99 ajoute cinq mots réservés au langage :

- **restrict**
- **inline**
- **_Complex**
- **_Imaginary**
- **_Bool**

III-B - Séquences d'échappement à deux caractères

Les séquences d'échappement à deux caractères permettent de remplacer certains caractères spéciaux qui peuvent être absent du clavier (comme les séquences d'échappement à trois caractères). Avant tout traitement, les lexèmes **<:**, **:>**, **<%**, **%>**, **%:** et **%:%:** sont respectivement remplacés par **[**, **]**, **{**, **}**, # et ##.

III-C - Mélange déclaration/code

En C90, les déclarations doivent être faites au début d'un bloc, alors qu'en C99 il est tout à fait possible d'écrire :

```
int main (void)
{
   printf ("Debut du code\n");
   int i = 10;
   printf ("i vaut %d\n", i);
   return 0;
}
```

Cependant ce genre de mélange n'est pas recommandé afin de préserver la lisibilité du code.

Cette fonctionnalité entraîne la possibilité de créer une variable de boucle temporaire pour l'instruction **for** :

```
for (int i = 0; i < 10; i++)
{
        /* variable i utilisable */
}
/* variable i inexistante */
```

Est équivalent à :

```
{
   int i;

   for (i = 0; i < 10; i++)
   {
      /* variable i utilisable */
   }
}
/* variable i inexistante */
```

Par conséquent, la variable *i* n'existera plus à la sortie de la boucle.

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/c99/

III-D - Initialisation des membres d'une structure

Il est possible d'initialiser les membres d'une structure, lors de sa déclaration, en utilisant leurs noms :

```
struct T
{
   int a;
   int b;
};

struct T s1 = {.a = 0, .b = 1};
/* ou dans le cas d'un tableau de structure : */
struct T s2[] = {[0].a = 0, [0].b = 0, [1].a = 1, [1].b = 1};
```

De plus, pour les structures de type **auto**, il est possible d'initialiser les membres avec des valeurs non connues au moment de la compilation :

```
int i = 0;
struct T s1 = {.a = i, .b = i+2};
```


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/c99/

IV - Sémantique

IV-A - Les fonctions en ligne

Le mot-clé **inline** est un nouvel attribut pour les fonctions qui ont pour but de remplacer le système de macro défini grâce à la commande du préprocesseur **#define**. Pour illustrer l'avantage de cette méthode par rapport aux macro, voici l'exemple classique du calcul du carré d'un nombre :

```
#define carre(a) ((a)*(a))
```

Si par malheur vous oubliez les parenthèses autour de la variable *a*, la macro peut produire un résultat inattendu, il faut aussi faire attention au effets de bord puisque la variable *a* est évaluée deux fois. Maintenant avec le mécanisme de fonction en ligne :

```
inline int carre (int a)
{
   return a*a;
}

int main (void)
{
   int b = carre (5);

   printf ("Le carre de 5 est %d\n", b);
   return 0;
}
```

Ceci supprime les dangers liés à l'utilisation de l'instruction **#define**. Les fonctions en ligne doivent être définies aux mêmes endroits que les macro classiques, ce qui peut alourdir les fichiers d'en-tête et surtout recopie la fonction dans plusieurs fichiers sources : pour éviter de se retrouver avec plusieurs fonctions globales portant le même nom, il est conseillé de toujours utiliser le mot-clé **inline** conjointement avec le mot-clé **static** (même dans un fichier d'en-tête).

> *L'attribut **inline** est une information d'optimisation pour le compilateur, si la mise en ligne n'est pas possible, le compilateur peut ignorer cet attribut.*

IV-B - long long

Le nouveau type d'entier **long long** (signé ou non) est représenté sur au moins 64 bits. Il s'accompagne des notation **LL** (ou **ll**) et **ULL** (ou **ull**).

IV-C - __func__

La variable __func__il s'agit d'une variable et non d'une macro comme *__FILE__* puisque le préprocesseur n'a pas la notion de fonction contient le nom de la fonction courante. Elle est déclarée ainsi :

```
static const char __func__[];
```

IV-D - Type de retour implicite

En C90, en l'absence de type de retour, le compilateur considère que cette fonction retourne un **int**, en C99 ce n'est plus possible : le type de retour doit être spécifié.

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/c99/

IV-E - Les pointeurs restreints

Le mot-clé **restrict** permet de spécifier qu'un pointeur est le seul à pointer sur une zone mémoire. Cela permet au compilateur d'opérer certaines optimisations. Par exemple, la fonction *memcpy* qui est utilisée pour copier deux zones de mémoires qui ne se chevauchent pas peut avoir comme prototype, en C99 :

```
void *memcpy (void *restrict dst, const void *restrict src, size_t size);
```

Le mot-clé ce place entre le caractère * et le nom du pointeur.

Comme pour les fonctions en ligne, cette directive peut être ignorée par le compilateur.

IV-F - Les tableaux de taille variable

Voilà une fonctionnalité qui va réjouir les réfractaires à l'utilisation des fonctions d'allocation dynamique. En effet, en C90 pour déclarer un tableau, sa taille doit être connue à la compilation. Le C99 propose un mécanisme nommé VLA (**V**ariable-**L**ength **A**rrays), qui permet de créer des tableaux dont la taille n'est connue qu'à l'exécution. Voici un exemple :

```
void foo (int n)
{
  int tab[n];
  ...
}
```

Comme toutes variables automatiques, elles sont détruites à la fin du bloc dans lequel elles sont déclarées (plus de fuites mémoires avec ce système), en contre partie l'allocation dynamique reste indispensable pour retourner l'adresse d'une zone mémoire.

On peut aussi déclarer un tableau de taille variable dans le prototype d'une fonction :

```
void foo (int n, int a[n]) { ... }
```

Il faut que la variable *n* soit définie avant, ce qui interdit d'écrire ceci :

```
void foo (int a[n], int n) { ... }
```

La déclaration d'une telle fonction peut s'écrire :

```
/* Declaration complete */
void foo (int n, int a[n]);
/* Declaration partielle */
void foo (int n, int a[*]);
```

Notez l'utilisation du caractère '' pour signaler un tableau de taille variable.*

Malgré le warning de gcc, le code fonctionne comme prévu.

Ce mécanisme très puissant a tout de même certaines limitations :

IV-G - Type de tableau incomplet

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/c99/

Lors de la déclaration d'une structure, il est possible de ne pas spécifier la taille d'un tableau, s'il s'agit du dernier membre de la structure :

```
struct incomplet
{
  int n;
  int tab[];
};
```

Cela permet de faire de belles choses comme ceci :

```
#include <stdio.h>
#include <stdlib.h>

struct incomplet
{
  int n;
  int tab[];
};

int main (void)
{
  struct incomplet *a = NULL;

  a = malloc (sizeof (*a) + 3 * sizeof (int));
  if (a)
  {
    a->n = 3;
    for (int i = 0; i < a->n; i++)
    {
      a->tab[i] = i;
    }
    printf ("a->tab = { ");
    for (int i = 0; i < a->n; i++)
    {
      printf ("%d, ", a->tab[i]);
    }
    printf (" };\n");
    free (a), a = NULL;
  }

  struct incomplet *b = NULL;

  b = malloc (sizeof (*b) + 5 * sizeof (int));
  if (b)
  {
    b->n = 5;
    for (int i = 0; i < b->n; i++)
    {
      b->tab[i] = i;
    }
    printf ("b->tab = { ");
    for (int i = 0; i < b->n; i++)
    {
      printf ("%d, ", b->tab[i]);
    }
    printf (" };\n");
    free (b), b = NULL;
  }
  return 0;
}
```

```
gege2061@debian:~/c99$ gcc -Wall -std=c99 main.c && ./a.out
a->tab = { 0, 1, 2,  };
b->tab = { 0, 1, 2, 3, 4,  };
```

Cela évite d'utiliser un pointeur pour *tab* et d'avoir une allocation supplémentaire à faire. Hélas il n'est pas possible de faire :

```
struct incomplet a = {3, 0, 1, 2};
struct incomplet b = {5, 0, 1, 2, 3, 4};
```


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/c99/

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/c99/

V - La bibliothèque standard

V-A - Les types entiers étendus

En C, la taille des types de variables n'est pas définie par la norme, seul une taille minimale est garantie. Le C99, dans le fichier d'en-tête *stdint.h* propose des types dont la taille est connue :

Types	Description
intN_t	Entier signé de N (8, 16, 32 ou 64Cette dernière taille n'est définie que pour les processeurs 64 bits) bits
uintN_t	Entier non signé de N (8, 16, 32 ou 64) bits
intmax_t	Entier signé de taille maximale
uintmax_t	Entier non signé de taille maximale
int_leastN_t	Entier signé d'au moins N (8, 16, 32 ou 64)bits
uint_leastN_t	Entier non signé d'au moins N (8, 16, 32 ou 64) bits
int_fastN_t	Entier rapide la taille est choisie afin d'augmenter la rapidité des opérations signé d'au moins N (8, 16, 32 ou 64) bits
uint_fastN_t	Entier rapide non signé d'au moins N (8, 16, 32 ou 64) bits

V-B - Les caractères étendus

En C90 il n'était pas possible d'utiliser de caractères spécifiques d'une langue (par exemple, les caractères accentués pour le français) et ce, même dans les commentaires. Pour pallier ce problème, la norme C99 propose un nouveau type de caractères étendus nommés **wchar_t** (pour *wide char*) défini dans *wchar.h*. Pour convertir un caractère étendu en entier, le type **int** peut ne pas être suffisant, à la place, il faut utiliser le type **wint_t**.

Toutes les fonctions de manipulation des chaînes de caractères (de la forme *str**) sont reprises sous la forme *wcs** (par exemple *wcslen*, *wcscmp* ou encore *wcscpy*) ainsi que les fonctions *wprintf* et *wscanf* pour l'affichage et la saisie formatée de chaîne de caractères étendues.

V-C - va_copy

Le fichier d'en-tête *stdarg.h* contient une nouvelle fonction *va_copy* qui permet de copier un objet de type *va_list*.

V-D - Le type booléen

Un booléen est un nouveau type de variable qui peut prendre uniquement la valeur vrai ou faux. La définition du type bool est faite dans le fichier d'en-tête *stdbool.h* et contient les définitions suivantes :

- **bool** : le type booléen
- **true** : la valeur vrai
- **false** : la valeur faux
- **__bool_true_false_are_defined** : si cette macro est définie, les valeurs **bool**, **true** et **false** le sont aussi.

*La définition du type **bool** a été placée dans un fichier d'en-tête pour garantir la compatibilité avec les anciens codes (il n'est pas rare de voir un code contenant déjà une définition de bool). Si, pour une raison quelconque, vous ne souhaitez pas inclure stdbool.h, le langage contient le type **_Bool** qui peut prendre les valeurs 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/c99/

(faux) et 1 (vrai).

V-E - Les nombres complexes

Les nombres complexes font leurs entrées dans le langage C grâce au type **complex** défini dans *complex.h*. Comme il existe trois représentations pour les nombres réels, il existe trois types de nombres complexes :

```
float complex
double complex
long double complex
```

Pour construire un nombre complexe, il suffit d'additionner sa partie réelle et sa partie imaginaire :

```
double partie_reelle = 2.0;
double partie_imaginaire = 3.0;
double complex z = partie_reelle + partie_imaginaire * I;
```

La macro *I* permet de construire la partie imaginaire.

Pour retrouver les parties réelles et imaginaires d'un complexe, on peut utiliser, respectivement les fonctions *creal* et *cimag* définies pour les trois type de complexes :

```
float crealf (float complex z);
double creal (double complex z);
long double creall (long double complex z);

float cimagf (float complex z);
double cimag (double complex z);
long double cimagl (long double complex z);
```

La bibliothèque standard du C99 propose aussi des fonctions pour calculer le module (*cabs**), l'argument (*carg**) et la représentation selon la projection sur la sphère de Riemann (*cproj**) d'un nombre complexe.

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/c99/

VI - Conclusion

Voilà, notre petit tour des nouveautés du C99 est terminé ! Je n'ai fait que vous présenter les nouveautés les plus importantes et sans approfondir, volontairement, certains points. Par exemple les nombres flottants ou les caractères étendus nécessiteraient un article entier, si le coeur vous en dit : http://club.developpez.com/redaction/ ;).

Cependant à l'heure actuelle, il n'est pas recommandé d'utiliser les spécificités de cette nouvelle norme puisqu'elle n'est pas encore intégralement implémentée dans tous les compilateurs (par exemple pour gcc : Status of C99 features in GCC). Il existe tout de même certains compilateurs conformes à la norme C99Certifié conforme par Perennial :

- Dinkum Unabridged Library (Dinkumware, Ltd)
- EDG C/C++ Compiler (Edison Design Group)
- Visual Age Compiler (IBM)
- LMPCC C99 Compiler (Lund Multiprocessor Compiler Company, AB)
- Sun Studio 9 (Sun Microsystems)

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/c99/

VII - Remerciements

Merci à fearyourself et à neguib pour leur relecture attentive 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/c99/