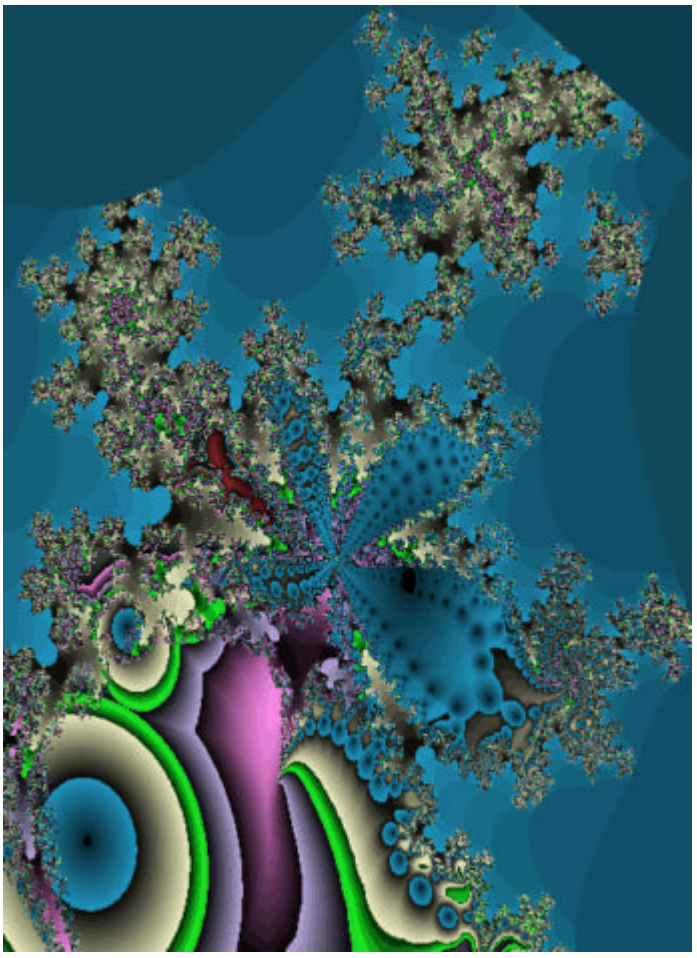
CHAPITRE 3

# *De C à C++*



C++ est un développement de C. Il garde la philosophie de C en y ajoutant des fonctionnalités, mais le principe de base de C qui veut que le programmeur est une personne responsable, qui sait ce qu'il fait, et que le langage ne doit en aucune manière restreindre la liberté du programmeur a été conservé. Tout au plus a-t-on (nettement) amelioré les possibilités de détection de problèmes potentiels. C++ ne peut de ce fait pas être envisagé totalement sans un regard en arrière vers C, de manière à comprendre la philosophie qui se cache derrière le langage.

C existe actuellement en deux variantes :

* C "classique", généralement désigné sous l'acronyme K&R C. L'acronyme signifie "Kernighan et Ritchie" (Brian Kernighan et Dennis Ritchie, les deux pères spirituels de C) et désigne un langage très proche du design originel de Dennis Ritchie.

*ANSI C (American National Standard Institute C). Version normalisée de C, désigne une variante nettement moins permissive que K&R C. Elle diffère de K&R C par le fait, entre autres, que le compilateur contrôle les passages de paramètre aux procédures, et par la déclaration des procédures (prototypes) avant l'utilisation.

## 3.1 *Des guerres de religion...*

C est souvent décrit comme *un langage qui combine toute l'élégance et la puissance du langage d'assemblage avec la lisibilité et la facilité de maintenance du langage assembleur*. Cette description est à interpréter en fonction de la personne qui l'utilise...

Il est intéressant de faire un petit retour en arrière, à l'époque où sévissait une impitoyable guerre de religion entre PASCAL et C, dans le plus pur style de "le langage que j'utilise est forcément le meilleur". Il s'agit d'un article reproduit dans "Comparing and Assessing Programming Languages" (Alan Feuer and Nerain Gehani, Prentice-Hall 1984) dont le titre était "Pourquoi PASCAL n'est pas mon langage favori". L'auteur de l'article n'était autre que Brian Kernighan. Cet article montre bien la différence fondamentale de philosophie entre PASCAL et C, bien que du point de vue de la structure, on puisse retrouver bien des notions qui semblent familières à première vue. Voici en substance un extrait de cet article (sa conclusion, en fait) :

> Il n'y a pas d'échappatoire.
>
> Ce dernier point est peut-être le plus important. Le langage est inadéquat mais limité, parcequ'il n'y a pas de moyen d'échapper à ses limitations. Il n'y a pas de coercion (casting) possible pour supprimer le contrôle de cohérence de types lorsque c'est nécessaire. Il n'y a pas de moyen de remplacer l'environnement "run-time" déficient par un autre à moins de contrôler les "procédures standard" implémentées dans le compilateur. Le langage est fermé.
>
> Les gens qui utilisent PASCAL pour faire de la programmation sérieuse tombent dans un piège fatal. Comme le langage est insuffisant, il doit être étendu. Mais chaque groupe étend le langage dans une direction différente, pour lui faire réfléter la personnalité du langage qu'ils désirent réellement. Il existe des extensions pour implémenter des modules de compilation séparée, des COMMON de type FORTRAN, des chaînes de caractères, des variables statiques internes, l'initialisation automatique, des nombres en octal, des manipulateurs binaires, etc...; tous ont pour objectif d'ajouter des fonctionnalités au langage dans un but déterminé, mais détruisent la portabilité par rapport aux autres groupes.
>
> Je pense que c'est une erreur d'utiliser PASCAL pour autre chose que son but original. Dans sa forme originale, PASCAL est un jouet, que l'on peut utiliser pour l'enseignement, mais pas pour la programmation dans un contexte réel.

Effectivement, PASCAL a actuellement été presque complétement remplacé par C dans les crénaux qu'il avait réussi à conquérir. Il garde encore une certaine popularité comme un langage pour programmeurs occasionnels dans le monde Windows, grâce à des outils remarquables comme Delphi, de Inprise. Rappelons encore que le système d'exploitation du Macintosh a été originellement écrit en PASCAL (Lisa PASCAL, une version passablement étendue du langage, avec déjà d'intéressantes extensions orientées objet). La critique de Kernighan peut s’étendre à tous les langages fermés, ne permettant pas aisément des extensions.

Par contraste, C est un langage très (trop?) permissif. A priori, il est possible de tout faire. D'ailleurs, le compilateur selon K&R ne contrôle même pas la validité des paramètres passés à une procédure. Il en serait d'ailleurs bien incapable, puisque les procédures ne doivent pas être déclarées avant leur utilisation (prototypes non nécessaires, voire interdits). L'exemple suivant est parfaitement licite en C K&R :

```
int      main(argc, argv)
int  argc;
char *argv[];
     {
     stupidProc(1, 2);
     }

int      stupidProc(r1)
double r1;

     {
     buggyProc("Dr Livingstone, I presume?");
     }

int      buggyProc(p1, p2, p3)
int      p1, p2, p3;
     {
     return(p1 + p2 + p3);
     }
```

Notons encore que l'exemple ci-dessus est typique des erreurs que peut occasionner l'utilisation de C selon K&R, car dans ce cas, le programme ne se plante pas forcément. Si la pile (*stack*) est ajustée en fonction de l'appel, il n'y a aucune raison que cette séquence d'instructions produise une erreur. Si la pile est ajustée en fonction du retour de la procédure, en revanche, le pointeur de pile sera faux après l'appel de `buggyProc()`, ce qui provoquera une erreur fatale au retour de `stupidProc()`. Remarquons encore que dans ce dernier cas, l'appel de `stupidProc()` lui-même peut ne pas conduire à une erreur, car il est possible que sur la machine utilisée, deux int équivalent à un double. L'exemple ci-dessus est par contre illégal en C ANSI, et par conséquent également en C++ qui reprend ANSI C tout en le consolidant.

Notons qu'il y a au moins autant (voire plus) d'articles incendiaires à l'égard de C qu'à l'égard de PASCAL. A commencer par Niklaus Wirth lui-même, qui affirmait que C n'était rien d'autre qu'un assembleur confortable. Mais ce cours n'a pas pour but de relancer une polémique qui n'a actuellement plus de raisons d'être, car réglée par les faits.

Actuellement, une très forte tendance à l'évolution de n'importe quel langage vers C++ se dessine. Cette évolution est dûe à plusieurs facteurs:

- La forte implantation de UNIX, qui est un produit indissolublement lié au langage C. Le langage de commande (shell scripts) lui-même présente de fortes similitudes avec la syntaxe de C.
- La forte compatibilité avec C, qui permet (?) un apprentissage plus aisé de C++ pour un programmeur C.
- La disponibilité, sur le marché, de nombreux produits d'infrastructure possédant des interfaces de programmation en C, comme par exemple OSF/MOTIF, Microsoft Foundation Classes (C++), Bedrock, stacks de protocoles OSI ou TCP/IP, gestionnaires de réseau de type OpenView, etc...
- Le support par C++, contrairement à C, de constructions modernes et puissantes, comme l'héritage, la surcharge d'opérateurs, l'encapsulation, etc...

- La promesse de l'apparition sur le marché de nombreuses bibliothèques (Motif 2.0, Interviews +, par exemple) possédant des interfaces de programmation en C++.
- La promesse de pouvoir réutiliser du code, et réduire ainsi les temps de développement, donc les coûts des produits logiciels.
- L'existence d'un standard ANSI pour C++, semblable dans sa finalité au standard défini pour ANSI C.
- Le fait que les principaux éditeurs de logiciels (Microsoft, par exemple, si l'on fait abstraction de Visual Basic) soutiennent cette évolution, du fait justement des infrastructures disponibles.

## 3.2 *Compilateurs C++ sur le marché*

C++ était prévu pour être utilisé en tant que préprocesseur générant du code C compilable ensuite par des compilateurs conformes à la norme ANSI. La complexité de C++ augmentant, de plus en plus de constructeurs ont choisi de développer des compilateurs C++ natifs. Les principales raisons en étaient la performance, la possibilité d'écrire des débogueurs spécifiques à C++, et l'espoir de pouvoir effectuer des optimisations plus poussées. Il n'existe actuellement que peu de compilateurs optimisants sur le marché, bien que presque tous se proclament comme tels.

Entre temps, le développement de C++ a échappé à AT&T, de même que le contrôle de UNIX. C'est la maison USL (UNIX Systems Lab) qui actuellement commercialise ces produits. USL est depuis 1993 une filiale de Novell, le géant des réseaux dans le monde PC. Novell a vraisemblablement senti dans cette alliance un moyen de parer aux tentatives[1] d'hégémonie de Microsoft (Windows95, Windows NT, Cairo, etc...). Le compilateur fourni par USL est toujours un préprocesseur (Cfront 3.5). Actuellement, USL travaille à la standardisation d'une librairie de base pour C++ (L++, Standard Components for C++), composant qui fait le plus gravement défaut à C++ dans sa version originelle. Signalons que des firmes rivales (HP entre autres, pour ne rien cacher au lecteur) poursuivent le même but avec une librairie appelée STL (Standard Template Library).

Sur le marché des ordinateurs de bas de gamme, les principaux compilateurs sont :

- Microsoft VISUAL C++. (Windows, Windows 95, Windows NT) Livré avec une librairie de classes, la Microsoft Foundation Class Library (MFC). Cette librairie n'a pratiquement aucune chance de standardisation par ANSI, vu son orientation Windows très nette. Par contre, il se pourrait que Microsoft parvienne à «standardiser» ANSI, vu sa toute puissance actuelle. Depuis la version 4.2, il permet de générer du code pour plusieurs plate-formes, et supporte l'interface Windows 32 bit.
- Inprise C/C++. (Windows) Contient également une bibliothèque de classes orientée vers les applications Windows. Compilateur très rapide et produisant un code de bonne qualité. Il existe une version "visuelle" nommée C++ Builder.
- Watcom C/C++. (DOS/Windows) Peut-être moins convivial que les deux premiers, mais nettement plus professionel. Grande compatibilité avec les compilateurs trouvés sous UNIX. Il existe également une version "visuelle" nommée Powersoft Optima C++.
- Symantec C++. Ce compilateur est supporté sur Macintosh et Windows, sans toutefois prétendre à une quelconque compatibilité entre les deux versions. Il s'agit d'un compilateur développé originellement par Zortech, mais complètement remanié.

1. Peut-on encore parler de tentatives quand on mentionne l'hégémonie d'un industriel en conjonction avec la firme dirigée par Bill Gates ?

- MPW C++. Ce compilateur a été longtemps uniquement disponible pour les développeurs "officiels" agrées Apple. Actuellement, il est assez facile à obtenir par Apple même.
- Metrowerk's Code Warrior. Compilateur très efficace, et capable de générer du code pour les processeurs Power PC, Motorola 680X0 et Intel. L'interface standard de Microsoft (Microsoft Foundation Classes) est également supporté. Pour les "explorateurs" de l'informatique, signalons que ce compilateur permet de générer du code également pour la Be Box de Jean-Louis Gassée (ex patron de Apple Computers), et pour Objective C (NextStep, futur Système 8 de Apple). Le même système comprend également un compilateur et une machine virtuelle Java.
- Enfin, pour les courageux ayant fait le pas UNIX sur PC, citons un excellent compilateur qui a de surcroît l'avantage non négligeable d'être gratuit : le compilateur gcc de GNU. Ce compilateur peut être obtenu dans divers sites sur Internet, et fonctionne sous BSD ou sous LINUX. L'ensemble LINUX / GNU peut être obtenu sur trois ou quatre (voire six ou sept) CD-ROM pour un prix inférieur à 100.- Le compilateur C++ de GNU, gcc (g++), a également été porté sur PC. Ce compilateur n'est pas à proprement parler recommandé aux débutants en C++, car les messages d'erreurs qu'il génère sont parfois pour le moins abscons... Ce compilateur a été récemment porté sur Win32 (Windows 95, Windows NT). Il est disponible chez www.cygnus.com.

## 3.3 *Notes sur la présentation de ce cours*

Ce cours sur C++ s'adresse en principe à des étudiants possédant déjà une certaine connaissance de C, ou au minimum une très bonne connaissance d'un langage structuré de type PASCAL. Ainsi, les exemples de certains chapitres font appel implicitement à des notions qui ne sont détaillées que plus loin dans le cours. Ceci est en particulier le cas pour les premiers chapitres.

Les étudiants ne possédant pas de notions préalables en C peuvent ainsi rencontrer quelques problèmes au cours de l'étude des chapitres concernant les types (standard, ou extensions de C++), les opérateurs standard et les instructions. Ces étudiants peuvent néanmoins aborder ce cours, sachant que certains exemples leur paraissant un peu obscurs seront explicités un peu plus avant dans le cours. A l'intention de ces étudiants, il existe aussi des exemples plus simples, ne faisant pas appel à des notions pas encore explicitées, et ils peuvent aussi aborder ces exemples en utilisant le contenu du chapitre "Le langage en un clin d'oeil", page32.

Les exercices proposés, en revanche, ne nécessitent généralement pas la connaissance de notions présentées plus loin dans le cours. Ceux-ci sont donc à la portée de tout le monde.