Cours Ada 95
pour le programmeur C++

V 1.0a (16/12/2004)

Auteur : Quentin Ochem (quentin.ochem@theatreux.org)

Note sur le présent document

Ce document a été rédigé pour permettre aux programmeurs connaissant le C++ d'apprendre rapidement les bases du langage Ada. De solides connaissances en C++ sont donc requises pour sa compréhension. Seul un tour d'horizon du langage Ada est proposé ici. Pour plus de détails, vous pouvez vous référer aux liens donnés dans le chapitre titré « références ».

Toutes les remarques ou suggestions sont largement appréciées. Vous pouvez les envoyer à cette adresse : quentin.ochem@theatreux.org.

1. Préface au lecteur sceptique

Parler de Ada et de C++, c'est un peu comme parler de Linux et de Windows, de Firefox et d'Internet Explorer, ou encore d'OpenOffice et de Word. On sombre vite dans une guerre de religion sans fin entre deux camps qui s'ignorent mutuellement et qui se plaisent à poser argument après argument. Nous éviterons donc dans ce document de tomber dans un fanatisme informatique, par ailleurs si tentant et si ludique, pour présenter Ada de la façon la plus neutre possible, en le comparant plutôt qu'en l'opposant à C++ (voir le chapitre « références »).

Mais pourquoi faire de l'Ada ? Pourquoi recommencer l'apprentissage d'un langage, par ailleurs réputé si austère, lorsque l'on maîtrise parfaitement un langage orienté objet comme C++ ? Ce C++ qui, entre autres avantages, est un standard connu et reconnu de tous ? Si vous lisez ces lignes, c'est que vous avez sans doute, au moins partiellement, répondu à la question. Passer de C++ à Ada, c'est faire un gros investissement. Bien que les paradigmes objets soient relativement proches entre ces deux langages, leur philosophie est complètement différente. En tant que débutant Ada, vous mettrez plus de temps à écrire votre code, et une fois écrit, vous mettrez encore plus de temps à lui faire passer la compilation. Vous aurez des difficultés structurelles à accéder à la mémoire bas niveau, vous ne pourrez pas faire de conversions implicites, vous serez contraints et forcés de mille manières différentes. Cerise sur le gâteau, vous ne parlerez pas le même langage que vos clients et concurrents (sauf si eux aussi travaillent en Ada), vous aurez donc des difficultés à réutiliser nombre de bibliothèques, vous aurez même des fonctions à interfacer vous-même. En Ada, pas de compilateur Borland, pas d'outil de compilation Microsoft, et de trop rares mentions sur les C.V. de vos recrues. Et pour couronner le tout, vos programmes compilés vous paraîtront énormes. Alors pourquoi Ada ?

Ada est un choix à faire sur le long terme, qui ne peut être rentable que s'il est assumé dans la durée. Il s'agit d'un langage en symbiose avec un certain nombre de concepts génie logiciel, souvent mal acceptés parce qu'ils imposent une réflexion plus poussée. Le code sera moins agréable à écrire, mais il gagnera en lisibilité. Il sera plus difficile à compiler mais une fois passé cette phase, il subira moins d'erreurs d'exécution. Il sera plus facile à maintenir, à réutiliser, et de très grosses erreurs auront été détectées tôt dans le développement là où, dans d'autres langages, elles peuvent parfois dormir des années… Pour exploser la veille de noël ou dix minutes avant une présentation. Passer à Ada ne sera pas forcément une partie de plaisir, mais ceux qui en ont fait l'effort affirment que leur efficacité a été décuplée. La question est : en avez-vous besoin ?

L'objectif de ce document n'est pas de vous apprendre à programmer en Ada. Pour étudier vraiment le langage, il faudrait au moins dix fois plus de pages. Mais il a pour but de vous en présenter les principales caractéristiques, vous permettant ainsi de réaliser facilement vos premiers programmes.

2. Généralités

Ada 95 est un langage qui implémente la quasi-totalité des notions que vous avez l'habitude d'utiliser en C++ : classes, héritage, polymorphisme, généricité. Les aficionados de Ada ont coutume de prétendre qu'il les implémente mieux, nous nous contenterons de dire qu'il les implémente différemment.

Tout le langage C++ est conçu pour permettre au programmeur de coder le plus facilement et de la façon la plus compacte possible, quitte à rendre le programme incompréhensible ou très difficilement débuggable en cas de problème. Cette tendance est actuellement contrebalancée par l'ajout de plus en plus systématique de warnings et de normes de codage. Mais personne n'est à l'abri d'une erreur d'inattention non contrôlée, et non vérifiable (cf critique du C++, en référence).

Le langage Ada part du principe exactement inverse. Le compilateur effectue de très nombreux tests qui bloquent le programme. Il rajoute même des tests à l'exécution (test de dépassements de tableaux par exemple). Le langage est très verbeux, on peut compter deux ou trois lignes Ada pour une ligne C++ en moyenne, et les mots clés sont explicites (begin et end au lieu des accolades ouvrantes et fermantes par exemple).

Contrairement à C++, Ada ne reconnaît pas la casse. Les variables VAR, var et VaR sont donc identiques. Ada accepte même les accents dans les identificateurs, bien que cette possibilité ne fasse pas l'unanimité. Certains craignent, entre autres, des incompatibilités entre des codes écrits en Ascii simple et en UTF-8.

Ada est voulu portable. En conséquence, les compilateurs subissent des tests poussés avant d'être autorisés à porter le label du langage, et n'ont qu'une marge très limitée pour étendre le langage, au travers de « directives de compilation », aussi appelées pragmas. En théorie, donc, lorsque vous écrivez un code pour un compilateur et que vous le recompilez sur un autre, il fonctionnera de la même façon. Si vous écrivez un programme pour un système, il fonctionnera pour un autre, sauf si vous faites des appels à des procédures exotiques, aux frontières du langage.

Ada est aussi rapide à l'exécution que C++, les programmes sont plus gros parce qu'ils rajoutent des tests, mais ces tests sont placés suffisamment intelligemment pour être effectués le plus rarement possible.