Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

Cloudy's Journey from FORTRAN to C, Why and How

G. J. Ferland

Physics, University of Kentucky, Lexington, KY 40506-0055

Abstract. Cloudy is a large-scale plasma simulation code that is widely used across the astronomical community as an aid in the interpretation of spectroscopic data. The cover of the ADAS VI book featured predictions of the code. The FORTRAN 77 source code has always been freely available on the Internet, contributing to its widespread use.

The coming of PCs and Linux has fundamentally changed the computing environment. Modern Fortran compilers (F90 and F95) are not freely available. A common-use code must be written in either FORTRAN 77 or C to be Open Source/GNU/Linux friendly. F77 has serious drawbacks - modern language constructs cannot be used, students do not have skills in this language, and it does not contribute to their future employability. It became clear that the code would have to be ported to C to have a viable future. I describe the approach I used to convert Cloudy from FORTRAN 77 with MILSPEC extensions to ANSI/ISO 89 C. Cloudy is now openly available as a C code, and will evolve to C++ as **gcc** and standard C++ mature. Cloudy looks to a bright future with a modern language.

1. Cloudy

The astronomical objects that produce the light we observe are seldom in thermodynamic equilibrium. This complication is why the spectrum is such a rich source of information. Most quantitative information, such as composition or dynamical state, is the result of the careful analysis of spectra. This analysis is best done by reference to complete numerical simulations of the emitting environment.

Cloudy is a large-scale plasma simulation code that fully simulates conditions in a cloud and predicts the resulting spectrum. The code is widely used across the astronomical community to produce roughly 100 papers per year. This wide use is possible because the code is platform independent and workstation friendly, in turn possible because it is close to ANSI standards.

Cloudy was born at the IOA Cambridge, in mid 1978, as a Fortran IV code. It evolved to become ~130,000 lines of FORTRAN 77 with MILSPEC extensions by mid-1996. That version is described in Ferland et al. (1998) and ADASS VI (Ferland et al. 1997).

I used a 1998-1999 sabbatical year at CITA, University of Toronto, to convert Cloudy from Fortran to C. This article describes why and how.

2. Why Convert to C?

There were three major reasons, listed in decreasing importance.

2.1. Job Market for Graduate Students

Most entering graduate students bring in some knowledge of C or Visual Basic. Most do not end up on a track leading to a tenured position at a research university, and many go into computer-related fields. The job market for C programmers is vastly richer than for Fortran experts. This is true both at the local level here in Lexington and in national astronomical centers. Students would be far more competitive in the job market if they had several years of experience developing large-scale C programs. Graduate students rely on the faculty to make choices that are in their long-term interests. If we can get our work done in a C environment, we owe it to our students to do so.

2.2. Open Source/GNU Friendly

Fortran 95 is a modern language. Unfortunately, the Open Source movement does not support Fortran beyond the **f2c** conversion utility and the **g77** compiler. Modern compilers are commercially available but are expensive, making modern Fortran more like IDL than a true ANSI language. As a result, portable Fortran code cannot go much beyond FORTRAN 77. At the same time, the C++ standard has now existed for well over a year and the **gcc** compiler and its standard template library are moving ever closer to full compliance. **gcc** has long been fully compliant with 1989 ANSI C.

2.3. Leverage Other Technologies

Most system shells and higher-level languages carry intellectual heritage from C. As universities change to better take advantage of the web, languages like Java, XGL, and SQL will become increasingly important. A C environment makes this both easy and natural.

3. Conversion Strategy

There were two immediate goals: Cloudy could not go out of scientific production for an extended time (it is totally supported by competitive extramural grants) and the effort must not break the code or introduce new bugs.

Extensive preparation was necessary, and was done without harming the original Fortran source. The variable name space was a major issue - Fortran does not have a global name space but uses common blocks for this purpose. Unique global names are necessary in C but not in Fortran. The first step was to insure unique and consistent names across the entire code.

Modern control structures such as enddo, break, and cycle, are not available in FORTRAN 77, but do exist as extensions to some compilers. The conversion from the F77 goto to these modern controls was done late in the initial process and resulted in a code that was not widely portable, but produced results that agreed with the original. After conversion this code was kept parallel with the C code to provide tests and comparisons.

Automatic conversion from Fortran to C was necessary to prevent the introduction of new bugs. The output from the C converter had to make sense to a human, and have the formatting that a human would have done. (This rules out **f2c**.) The resulting source also had to be freely redistributable on the Internet and run on all platforms that had an ANSI C compiler. This meant that the source for any helper routines also had to be open. The **forc** program from Cobalt Blue (http://www.cobalt-blue.com) was the only conversion routine that fulfilled these requirements. I know of no conversion utility for F90 or F95, this had to take place from a source close to F77.

4. Post-Conversion Issues

The conversion process produced a C code that could be compiled without errors and produced the same results as the original Fortran code. Next came a series of corrections that had to be made to the translated source, largely due to the different natures of the languages.

4.1. C Arrays Start From 0, and Have no Bounds Checking

Perhaps the biggest single deficiency in C is the lack of any standard bounds checking on array indices. This is a fundamental limitation due to the way arrays are declared - as pointers - in routines that access them. This is a problem because exceeded array bounds are a common mistake and hard to detect.

The C array counting scheme is a second problem. Fortran counts an N-dimensional array from 1 to N, while C counts from 0 to N-1. **forc** converted this in a reliable way that was not a pretty sight – it left the original limits on loops but subtracted 1 from all array references.

Unfortunately, the C array counting scheme does not make sense in a physics code. Hydrogen will always be element number 1, carbon 6, and iron 26. C's off-by-one addressing was a great chance for confusion and bugs.

The array addressing was changed back to the FORTRAN style, and two additional array elements, at 0 and N+1, were also allocated (memory is cheap today). These extra elements were set to NaN to provide an "electric fence" to ensure that out-of-bounds elements are never used. This made more physical sense and provided an automatic and fast means of bounds checking.

4.2. IO Problems

IO is fundamentally different in the two languages. Fortran is line-based, being designed for line printers and card readers, while C is character-based, being designed for terminals. The translated code provided an infrastructure that fully simulated the Fortran environment in the C code. All of this was rewritten to take advantage of the C environment. Today, only native C functions are used for IO.

4.3. Block Data

Large quantities of physical constants are naturally stored in "block data" routines in Fortran. This concept does not exist in C or other modern languages, the preferred style being to gather this data from ancillary files. **forc** translated a block data into large routines that were executed to set variables to values. In some cases these could be tens of thousands of lines long, and they could not be compiled with **gcc** and moderate levels of optimization. The original block data routines were recoded into the C method of reading ancillary files.

4.4. Other Stylistic Differences

This discussion gives a hint of the basic differences between these two languages. There are many others that pose stylistic, but not fundamental, problems. Portions of the converted code simply do not look like good C code - it looks like converted Fortran. This converted code works well and does get the job done efficiently. Converting it to the C way of doing things has become a continuing part-time effort. It is done slowly on a case-by-case basis as routines are improved or changed.

5. Final Results, and Some Observations

The C version of Cloudy has been released on the web and is now ~160,000 lines of ANSI 89 C (http://www.pa.uky.edu/~gary/cloudy/). There is also a more extensive set of notes on the conversion process (http://nimbus.pa.uky.edu/ cfromfortran/). Some general observations follow.

The C code is slightly faster than the Fortran version. This is mostly the result of a general cleanup of the code's kernel rather than differences between the two languages. This is also contrary to rumors of loss in speed for scientific calculations in C.

One striking difference is the fact that, on the average Unix box, the C development environment is better than the Fortran. This includes the many types of lint, source level debuggers, integrated development environments, and peer support, and reflects the fact that the OS itself is a C code.

The feedback from the user community has been largely positive. Cloudy is mostly used by graduate students who were told to do so by their advisor. The C environment is natural to these young people.

Cloudy is now "clean C", meaning that the files can be renamed to *.cpp and then built as a C++ program. The code will move to C++ as **gcc** and its STL mature. This will begin with the next major update to the code.

Acknowledgments. The development of Cloudy is supported by NSF and NASA. Peter Martin and Dick Bond provided the atmosphere at CITA to do this work. I thank Anuj Sarma for his comments.

References

Ferland, G. J., Korista, K. T., & Verner, D. A. 1997, in ASP Conf. Ser., Vol. 125, Astronomical Data Analysis Software and Systems VI, ed. G. Hunt & H. E. Payne (San Francisco: ASP)

Ferland, G. J., Korista, K. T., Verner, D. A., Ferguson, J. W., Kingdon, J. B., & Verner, E. M. 1998, PASP, 110, 761