Chapter I

Introduction

1.1 Background and Rationale

Computer programmers spend a lot of time writing programs that do simple, mechanical data manipulation such as changing the format of data, checking its validity, finding items with some property, summing up numbers, printing reports, and the like. Although most of these jobs are not complicated, it is a real annoyance to have to write special purpose programs in conventional languages like COBOL or C each time such a task comes up.

The AWK language, first implemented on the Unix System in 1977, is designed to make it possible to handle such tasks with ease and convenience. Because so many things are automatic in AWK - input, field splitting, pattern matching, storage management, initialization - AWK programs are usually much smaller than they would be in a more conventional language, thus saving a lot of programmers' valuable time and labor. As a result, AWK rapidly became very popular and the AWK language processor has become one of the standard programming tools on every version of Unix since 1979, and later on has been implemented on other operating systems such as MSDOS and VMS as well.

AWK language processors have traditionally been implemented as interpreters, making program development convenient and fast. However, this implementation scheme also means that AWK programs will typically run much slower than equivalent programs written in a compiled language. The performance could be increasingly intolerable as a working AWK program is applied to bigger and bigger data files. Thus, for tasks where run time is more important than program development time, programmers may have to go back to more

conventional, compiled languages like COBOL or C and bear with all their burdensome syntactic baggage.

In order to help lessen this run-time performance problem, a new processing scheme for the AWK language, the translation of AWK programs into equivalent C programs has been designed and developed. An AWK-to-C translator program and its supporting tools have been constructed for this purpose. With the translator at his or her disposal, the programmer could continue to enjoy all the conveniences of writing programs in AWK and then should the performance of the interpreted program prove too slow for the task at hand, he or she could use the translator to translate the AWK program into an equivalent C program and then compile it using the system's existing C compiler to produce a readily executable program that would run faster than the original interpreted AWK program.

In addition to improving the performance of AWK programs, the AWK-to-C translator has another important use. On the system without an AWK language processor but has a C compiler available, the AWK-to-C translator ported to this system, together with the existing C compiler constitute a complete implementation of the AWK language for the system. In other words, the system could then process any AWK program despite the absence of a "real" AWK processor.

Therefore, the author believes that the AWK-to-C translator would serve its purposes well and become one of the useful software tools frequently used by AWK programmers.

1.2 Objectives

The objective of the study is to design and develop a complete software system for translating any AWK program into a functionally equivalent C program. The software system includes an AWK-to-C translator and a run-time library of subroutines that support the execution of the generated C program.

1.3 Scope and Limitations

The design and development of the AWK-to-C translation system are subjected to the following scope and limitations:

1. The AWK language that the translator recognizes is the one defined and described in Aho, Kernighan, and Weinberger (1988)*.

2. The C language that the translator used for the generated programs is the one defined by the 1989 ANSI standard and described in Kernighan and Richie (1988).

3. The entire AWK-to-C translation system was developed on the AT&T Unix System V Release 4.0 and has only been tested successfully on that system. Since many of the Unix-specific language development tools and library routines was used, the source code could hardly be ported to other operating systems without omitting some language features. However, the author has made an effort to make it reasonably portable among the versions of Unix that have an ANSI C compiler available.

1.4 Development Procedure

Development of the AWK-to-C translator follows the following steps:

1. Study the syntax and semantics of the AWK language.

2. Write a complete grammar for the AWK language and check its correctness by constructing an AWK language recognizer program based on this grammar. The only function of this program is to check the syntactic correctness of AWK programs.

---

* This version of language definition is also recognized by the AWK interpreter *nawk*, which has been bundled with every version of the Unix System V since Release 3.1.

3. Design and develop a prototype version of the AWK-to-C translation system that recognizes a small, yet nontrivial subset of the AWK language. The main purpose of the prototype is to gain some experiences and to experiment with various ways of implementing many parts of the AWK language.

4. Design the run-time organization of the translator-generated programs.

5. Design and develop the production version of the translation system based on the grammar written in step 3 and using the run-time organization designed in step 4.

6. Test and debug the translation system.

7. Do the execution-time performance comparison between the AWK source programs running with the existing AWK interpreter on the system and their corresponding translator-generated C programs.

8. Assess the development results, draw conclusions, and write the thesis documentation.

1.5 Expected Benefits and Applications

The usefulness of the AWK-to-C translator is twofold:

1. The AWK-to-C translator serves as an alternative way of processing the AWK language, which is generally faster than using the traditional AWK interpreter to run AWK programs.

2. On the systems with a C compiler but without an AWK language processor, the AWK-to-C translation system, together with the system's C compiler, also serves as a complete implementation of the AWK language on the system. Such systems could run AWK programs in two ways. First, the whole AWK-to-C translation system could be ported to the system and would then be used for translating AWK

programs into C programs, which, in turn, would be compiled by the C compiler and run. Or, secondly, the cross-translation method could be employed; that is, the translator installed on another system could be used for translating an AWK program and the resulting C program would then be ported to the target system to be compiled and run there.

1.6 Literature Review

1.6.1 AWK Language

The latest version of the AWK language is defined and described in Aho, Kernighan, and Weinberger (1988). Chapter 2 of this book covers the entire language in a concise, systematic manner. The rest of the book contains a wide variety of examples, showing the breadth of applicability of AWK.

1.6.2 C Language

Kernighan and Ritchie (1988), the classic book on C, presents the entire ANSI C language in a clear, brief manner. The appendices also provide a concise language reference manual and a summary of the facilities of the C standard library.

1.6.3 Programming Language Translation

A good description of the basic theory underlying the implementation of programming languages may be found in Pratt (1984). Chapter 2 outlines the basic approaches to language implementation. Chapter 3 through 8 describes fundamental language constructs and data objects, and how to implement them on a conventional computer system.

An in-depth introduction to the basic structure of a compiler/translator and how its components work together can be found in Chapter 1 and 2 of Aho, Sethi, and Ullman (1986). A good discussion on symbol-table handling techniques is given in chapter 8 of Tremblay and Sorenson (1985).

1.6.4 Unix Language Development Tools

The Unix programs *lex* and *yacc* was used to develop the scanner and the parser modules of the AWK-to-C translator. Lex was originally written by M. E. Lesk and was described in Lesk and Schmidt (1979) while yacc was developed by S. C. Johnson and was described in Johnson (1979). An excellent description and examples of how to use lex and yacc together to develop a language processor is in Friedman and Schriener (1985).

Kernighan and Pike (1984) provides excellent detailed accounts on how to program in the Unix environment. Chapter 8 of this book also covers a development of a programmable calculator using lex and yacc, providing many insights into how a language processor really works.