Chapter 2

An Overview of ANSI C

This chapter covers the following topics:

- "What Is ANSI C?" briefly discusses the scope of the new standard.
- "Helpful Programming Hints" lists some programming practices to avoid and some to use.
- "Areas of Major Change" lists the major changes to C made by the ANSI standard.

What Is ANSI C?

The ANSI standard on the programming language C is designed to promote the portability of C programs among a variety of data–processing systems. To accomplish this, the standard covers three major areas: the environment in which the program compiles and executes, the semantics and syntax of the language, and the content and semantics of a set of library routines and header files. *Strictly conforming programs* are programs that:

- use only those features of the language defined in the standard
- do not produce output dependent on any ill–defined behavior
- do not exceed any minimum limit.

Ill–defined behavior includes *implementation–defined, undefined,* and *unspecified* behavior. The term refers to areas that the standard does not specify.

This ANSI C environment is designed to be, in the words of the standard, a *conforming hosted implementation,* which is guaranteed to accept any *strictly conforming program.* Extensions are allowed, as long as the behavior of strictly conforming programs is not altered.

Besides knowing which features of the language and library you may rely on when writing portable programs, you must be able to avoid naming conflicts with support routines used for the implementation of the library. To avoid such naming conflicts, ANSI divides the space of available names into a set reserved for the user and a set reserved for the implementation. Any name that does not begin with an underscore and is neither a keyword in the language nor reserved for the ANSI library, is in the user's namespace. (This rule is given for simplicity. The space of names reserved for the user is actually somewhat larger than this.)

Strictly conforming programs may not define any names unless they are in the user's namespace. New keywords as well as those names reserved for the ANSI library are discussed in "Standard Headers".

Compiling ANSI Programs

To provide the portable clean environment dictated by ANSI while retaining the many extensions available to Silicon Graphics users, two modes of compilation are provided for ANSI programs. Each of these modes invokes the ANSI compiler and is selected by a switch to *cc*(1):

-ansi enforces a pure ANSI environment, eliminating Silicon Graphics extensions. The

ANSI symbol indicating a pure environment (*__STDC__*) is defined to be 1 for the preprocessor. Use this mode when compiling *strictly conforming programs,* as it guarantees purity of the ANSI namespace.

-xansi adds Silicon Graphics extensions to the environment. This mode is the default. The ANSI preprocessor symbol (*__STDC__*) is defined to be 1. The symbol to include extensions from standard headers (*__EXTENSIONS__*) is also defined, as is the symbol to inline certain library routines that are directly supported by the hardware (*__INLINE_INTRINSICS.*) Note that when these library routines are made to be intrinsic, they may no longer be strictly ANSI conforming (e.g., *errno* may not be set correctly).

Some key facts to keep in mind when you use ANSI C are listed below:

- Use *only* **-lc** and/or **-lm** to specify the C and/or math libraries. These switches ensure the incorporation of the ANSI version of these libraries.
- The default compilation mode is shared and the libraries are shared.
- Use the switch **-fullwarn** to receive additional diagnostic warnings that are suppressed by default. Silicon Graphics recommends using this option with the **-woff** option to remove selected warnings during software development.
- Use the switch **-wlint** (**-32** mode only) to get lint−like warnings about the compiled source. This option provides lint−like warnings for ANSI and **-cckr** modes and can be used together with the other *cc*(1) options and switches.

If you want to compile code using traditional C (that is, non−ANSI), use the switch **-cckr**. The dialect of C invoked by **-cckr** is referred to interchangeably as **-cckr**, "the previous version of Silicon Graphics C," and "traditional C" in the remainder of this document.

You can find complete information concerning ANSI and non−ANSI compilation modes in the online manual page for *cc*(1).

Helpful Programming Hints

Although the ANSI Standard has added only a few new features to the C language, it has tightened the semantics of many areas. In some cases, constructs were removed that were ambiguous, no longer used, or obvious hacks. The next two sections give two lists of programming practices. The first section recommends practices that you can use to ease your transition to this new environment. The second section below lists common C coding practices that cause problems when you use ANSI C.

Recommended Practices

Follow these recommendations as you code:

- Always use the appropriate header file when declaring standard external functions. Avoid embedding the declaration in your code. Thus you avoid inconsistent declarations for the same function.
- Always use function prototypes, and write your function prologues in function prototype form.

- Use the *offsetof()* macro to derive structure member offsets. The *offsetof()* macro is in <*stddef.h*>.
- Always use casts when converting.
- Be strict with your use of qualified objects, such as with **volatile** and **const**. Assign the addresses of these objects only to pointers that are so qualified.
- Return a value from all return points of all non–**void** functions.
- Use only structure designators of the appropriate type as the structure designator in `.` and `->` expressions (that is, ensure that the right side is a member of the structure on the left side).
- Always specify the types of integer bitfields as **signed** or **unsigned**.

Practices to Avoid

Avoid these dangerous practices:

- Never mix prototyped and nonprototyped declarations of the same function.
- Never call a function before it has been declared. This may lead to an incompatible implicit declaration for the function. In particular, this is unlikely to work for prototyped functions that take a variable number of arguments.
- Never rely on the order in which arguments are evaluated. For example, what is the result of the code fragment `foo(a++,a,`...)?
- Avoid using expressions with side effects as arguments to a function
- Avoid two side effects to the same data location between two successive sequence points (for example, `x=++x;`).
- Avoid declaring functions in a local context, especially if they have prototypes.
- Never access parameters that are not specified in the argument list unless using the **stdarg** facilities. Use the **stdarg** facilities only on a function with an unbounded argument list (that is, an argument list terminated with …).
- Never cast a pointer type to anything other than another pointer type or an integral type of the same size (unsigned long), and vice versa. Use a union type to access the bit–pattern of a pointer as a nonintegral and nonpointer type (that is, as an array of chars).
- Don't hack preprocessor tokens (for example, `FOO/**/BAR`).
- Never modify a string literal.
- Don't rely on search rules to locate *include* files that you specify with quotes.

Areas of Major Change

Major changes to C made by the ANSI standard include:

- Some *preprocessor changes* are noteworthy. The changes are in practices that, although questionable, are not uncommon.

- Rules for *disambiguating names* have been more clearly defined. Most of these changes allow greater freedom to use the same name in different contexts.
- *Types* have undergone some significant changes in the areas of *promotions* and more strictly enforced *compatibility* rules. In addition, the compiler is more strict about mixing *qualified* and *unqualified* types and their pointers.
- *Function prototypes* are more completely observed. Many warnings concerning prototypes in traditional C are now errors under ANSI.
- A few external names have been changed for conformance.