The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

5.1.2 Execution environments

Two execution environments are defined: *freestanding* and *hosted*. 149

Commentary

Freestanding is often referred as an *embedded system*, outside of the C Standard's world.

Other Languages

Most languages are defined (often implicitly) to operate in a hosted environment. The Java environment has a single, fully specified environment, which must always be provided.

Common Implementations

Vendors tend to sell their products into one of the two environments. The `gcc` implementation has been targeted to both environments.

program startup

In both cases, *program startup* occurs when a designated C function is called by the execution environment. 150

Commentary

Executing the developer-written functions is only part of the process of executing a program. There are also various initialization and termination activities (discussed later).

The C Standard does not deal with situations where there may be more than one program executing at the same time. Its model is that of a single program executing a single thread of execution.

Common Implementations

Implementations use one of two techniques to call the designated function (called `main` here):

```
result_code=main(argc, argv);
```

or,

```
exit(main(argc, argv));
```

The method used to make the call does not affect startup, but it can affect what happens when control is handed back to the startup function by the program.

static storage duration initialized before startup

All objects with static storage duration shall be *initialized* (set to their initial values) before program startup. 151

Commentary

This is a requirement on the implementation.

static storage duration
object initialized but not explicitly

This initialization applies to all objects having file scope and objects in block scope that have internal linkage. The initial values may have been provided explicitly by the developer or implicitly by the implementation. Once set, these objects are never reinitialized again during the current program invocation, even if `main` is called recursively (permitted in C90, but not in C99 or C++).

C++

In C++ the storage occupied by any object of static storage duration is first zero-initialized at program startup (3.6.2p1, 8.5), before any other initialization takes place. The storage is then initialized by any nonzero values. C++ permits static storage duration objects to be initialized using nonconstant values (not supported in C). The order of initialization is the textual order of the definitions in the source code, within a single translation unit. However, there is no defined order across translation units. Because C requires the values used to initialize objects of static storage duration to be constant, there are no initializer ordering dependencies.

Other Languages

Some languages (e.g., the original Pascal Standard) do not support any mechanism for providing initial values to objects having static storage duration. Java allows initialization to be delayed until the first active use of the class or interface type. The order of initialization is the textual order of the definitions in the source code.

Common Implementations

Using the as-if rule to delay initialization until the point of first reference often incurs a high execution-time performance penalty. So most implementations perform the initialization, as specified, prior to startup.

In freestanding environments where there are warm resets, realtime constraints on startup and other constraints sometimes mean that this initialization is omitted on startup. A program can rely on objects being explicitly assigned values through executed statements (appearing in the source code).

The IAR PICmicro compiler[1] supports the `__no_init` specifier which causes the translator to not generate machine code to initialize the so-designated objects on program startup.

Coding Guidelines

use initializers discussion

initialization syntax

It is guaranteed that all objects having file scope will have been initialized to some value prior to program startup. Some coding guideline documents require that all initializations be explicit in the source code and implicit initialization not be relied on. However, it is unclear what the proposers of such a guideline recommendation are trying to achieve. Experience suggests that these authors are applying rationales whose cost/benefit is only worthwhile for objects defined in block scope.

---

152 The manner and timing of such initialization are otherwise unspecified.

initialization timing

Commentary

initializer static storage duration object

For objects defined at file scope the standard only supports initialization expressions whose value is known at translation-time. This means that there are no dependencies on the order of these initializations. It is not possible to write a conforming C program that can deduce the manner and timing of initialization.

Floating-point constants, the representation of which can vary between hosts, may be stored in a canonical form in the program image, and be converted by startup code to the representation used by the host.

Common Implementations

linkers

The manner in which initialization occurs can take several forms. The program image may simply contain a list of values that need to be copied into memory; many linkers merge the initializations of sequences of contiguous memory locations into a single block, this exact memory image being copied on startup. For large blocks of memory being initialized to a single value, translators may generate code that loops over those locations, explicitly setting them. Some host environments set the memory they allocate to programs to zero, removing the need for much initialization on behalf of the developer, which for large numbers of locations is zero.

Addresses of objects that have been assigned to pointers may have to be fixed up during this initialization (information will have been stored in the program image by the linker).

In a freestanding environment values may be held in ROM, ready for use on program startup.

Coding Guidelines

Use of the C++ functionality of having nonconstant expressions for the initializers of static storage duration objects may be intended or accidental. Using this construct intentionally is making use of an extension. It may also be necessary to be concerned with order of initialization issues (the order is defined by the C++ Standard).

The use may be accidental because some C++ compilers do not diagnose such usage when running in their C mode. Checking the behavior of the translator using a test case is straight-forward. The action taken, if any, in those cases where the use is not diagnosed will depend on the cost of independent checking of the source (via some other tool, even a C translator) and the benefit obtained. This is a management, not a coding guidelines, issue.

Example

```
extern int glob_1;

```

```
static int sglob = glob_1; /* not conforming */
                            // conforming
```

Program termination returns control to the execution environment. 153

Commentary

freestanding environment program termination

There are many implications hidden beneath this simple statement. Just because control has been returned to the execution environment does not mean that all visible side effects of executing that program are complete. For instance, it may take some time for the output written to files to appear in those files.

Program termination also causes all open files to be closed.

C++

3.6.1p1 *[Note: in a freestanding environment, start-up and termination is implementation defined;*

3.6.1p5 *A **`return`** statement in `main` has the effect of leaving the main function (destroying . . . duration) and calling `exit` with the return value as the argument.*

18.3p8 *The function `exit()` has additional behavior in this International Standard:*

. . . Finally, control is returned to the host environment.

Common Implementations

In a hosted environment program, termination may, or may not, have a large effect on the execution environment. If the host does not support concurrent execution of programs, termination of one program allows another to be executed.

In a hosted environment the call to the designated function, when startup occurs, usually makes use of a stack that has already been created by the host environment as part of its process of getting a program image ready to execute. Program termination is the point at which this designated function returns (or a call to one of the library functions `exit`, `abort`, or `_Exit` causes the implementation to unwind this stack internally). Control then returns to the code that performed the original invocation.

Forward references: storage durations of objects (6.2.4), initialization (6.7.8). 154

References

1. IAR Systems. *PICmicro C Compiler: Programming Guide*, iccpic-1 edition, 1998.