Glen McCluskey
Mentor Graphics
March 2, 1990

INTRODUCTION

This position paper describes a proposed chapter in the standard on C++ environment issues. It presents only a detailed outline, for later filling in. Much of the paper is based on experience with using the proposed features in a large industrial setting.

There are several reasons for splitting out this section and expanding it. At present, the information is scattered through the reference manual, and collecting it would be useful.

The need for expansion is driven by several considerations. For example, my company, and based on my experience other companies doing large-scale development, have a need to interface C++ with other languages, to use a variety of shared library and dynamic linking schemes, etc. Some of this type of thing is implementation dependent, while large other parts of it could be standardized. It is better to standardize as much as possible, rather than have every implementation do it differently.

Also, there is much interest in programming environments for C++. If possible, the committee should standardize hooks, such as vtbl layout and naming, to facilitate such environments.

PROGRAM INVOCATION

1. Exactly one "main" in a program, with C linkage.
2. Argc / argv usage.
3. Envp usage.
4. Reentrancy - having a guard to allow main() to call itself.

PROGRAM TERMINATION

1. Use of exit() and _exit().
2. Use of abort() and its effect on static destructors being called.
3. Use of atexit().
4. Doing "return" from "main" and calling of static destructors.
5. Reentrancy of exit() for static destructors.

RUNTIME PROGRAM LAYOUT

1. Use of text, data, and bss sections.

2. Zeroing of data and bss.

3. Whether C++ really has a bss section, given that global objects are initialized to prevent multiple definitions.

STATIC CONSTRUCTORS AND DESTRUCTORS

1. LIFO order between constructors and destructors.

2. Use and desirability of allocation hooks.

3. Patch linking.

4. Patch linking and its interaction with relocated start addresses for programs and dynamic loading.

5. Schemes for specifying static constructor ordering.

INTERFACING WITH OTHER LANGUAGES

1. Use of _cplusplus_start() and _cplusplus_finish() functions to call static constructors and destructors from other languages.

2. Use of _main() with other languages.

3. What if main() is not in C?

4. Reentrancy guards on interface functions such as _main().

5. Guarding overloaded exit() against the case that it was called without _main() being called first.

6. Use of asm().

PROGRAM STRUCTURE

1. Separate compilation.

LINKING

1. Type-safe linkage.

2. Enforcing only one definition of an object.

3. Linkage specifications.

NAMING

1. Use of 'extern "C"'.

2. What to do about temporary names in function signatures.

DYNAMIC LOADING

1. Calling static constructors and destructors.

2. Relocation and patch.

3. Provision of a nameof() operator.

4. Need to standardize vtbl and static constructor naming and protocols.