The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones
derek@knosof.co.uk

Copyright ©2002-2008 Derek M. Jones. All rights reserved.

5.1.2.2.1 Program startup

hosted environment startup

The function called at program startup is named **main**. 162

Commentary

That is, the user-defined function called at program startup. This function must have external linkage (it is possible for a translation unit to declare a function or object, called `main`, with internal linkage). There is no requirement that the function appear in a source file having a specific name.

Other Languages

A Java Machine starts execution by invoking the method main of some specified class, . . .

In Fortran and Pascal the function called on startup is defined using the keyword **PROGRAM**.

Common Implementations

Implementations use a variety of techniques to start executing a program image. These usually involve executing some internal, implementation-specific, function before `main` is called. The association between this internal function and `main` is usually made at link-time by having the internal function make a call to `main`, which is resolved by the linker in the same way as other function calls.

Coding Guidelines

Having the function `main` defined in a source file whose name is the same as the name of the executable program is a commonly seen convention.

The implementation declares no prototype for this function. 163

Commentary

There is no declaration of `main` in any system header, or internally within the translator. For this reason translators rarely check that the definition of `main` follows one of the two specifications given for it.

Common Implementations

A function named `main` is usually treated like any other developer-defined function.

Coding Guidelines

Because the implementation is not required to define a prototype for `main` no consistency checks need be performed against the definition of `main` written by the developer.

It shall be defined with a return type of **int** and with no parameters: 164

```
int main(void) { /* ... */ }
```

Commentary

Many lazy developers use the form:

```
1  main() { /* ...  */ }
```

declaration at least one type specifier

C99 does not allow declaration specifiers to be omitted; a return type must now be specified.

Usage

There was not a sufficiently large number of instances of `main` in the `.c` files to provide a reliable measure of the different ways this function is declared.

165 or with two parameters (referred to here as **argc** and **argv**, though any names may be used, as they are local to the function in which they are declared):

main prototype

```
int main(int argc, char *argv[]) { /* ... */ }
```

Commentary

The parameters of `main` are treated just like the parameters in any other developer-defined function.

Other Languages

. . . , passing it a single argument, which is an array of strings.

Java

```
class Test {
   public static void main(String[] args) {
      /* ... */
      }
   }
```

Coding Guidelines

The identifiers `argc` and `argv` are commonly used by developers as the names of the parameters to `main` (the standard does not require this usage).

166 or equivalent;[9)]

Commentary

Many developers use the form:

```
int main(int argc, char **argv)
```

on the basis that arrays are converted to pointers to their element type.

C++

The C++ Standard gives no such explicit permission.

167 or in some other implementation-defined manner.

Commentary

The Committee recognized that there is additional information that a host might need to pass to `main` on startup, but they did not want to mandate anything specific. The following invocations are often seen in source code:

```
void main()
```

and

```
int main(argc, argv, envp)
```

C90

Support for this latitude is new in C99.

C++

The C++ Standard explicitly gives permission for an implementation to define this function using different parameter types, but it specifies that the return type is **int**.

3.6.1p2

It shall have a return type of **int**, *but otherwise its type is implementation-defined.*

. . .

[Note: it is recommended that any further (optional) parameters be added after `argv`. *]*

Common Implementations

In a wide character hosted environment, `main` may be defined as:

```
int main (int argc, wchar_t *argv[])
```

Some implementations[1] support a definition of the form:

```
int main (int argc, char *argv[], char *env[])
```

where the third parameter is an array environment strings such as `"PATH=/home/derek"`.

The POSIX `execv` series of functions pass this kind of additional information to the invoked program. However, `main` is still defined to take two parameters for these functions; the environment information is assigned to an external object `extern char **environ`.

Coding Guidelines

Programs that need to make use of the implementation-defined functionality will obviously define `main` appropriately. If it is necessary to access environmental information, it is best done through use of implementation-supported functionality.

If they are declared, the parameters to the **main** function shall obey the following constraints: 168

Commentary

This is a list of requirements on implementations. It lists the properties that the values of `argc` and `argv` can be relied on, by a program, to possess. The standard does not define any constraints on the values of any, implementation-defined, additional parameters to `main`.

Common Implementations

The constraints have proved to be sufficiently broad to be implementable on a wide range of hosts.

Coding Guidelines

These constraints are the minimum specification that can be relied on to be supported by an implementation.

— The value of **argc** shall be nonnegative. 169

Commentary

The value of `argc`, as set by the implementation on startup is nonnegative. The parameter has type **int** and could be set to a negative value within the program.

Common Implementations

Most hosts have a limit on the maximum number of characters that may be passed to a program on startup (via, for instance, the command line). This limit is also likely to be a maximum upper limit on the value of `argc`.

Coding Guidelines

Knowing that `argc` is always set to a nonnegative value, a developer reading the code might expect it to always have a nonnegative value. Assigning a negative value to `argc` as a flag to indicate a special condition is sometimes seen. The extent to which this might be considered poor practice because of violated developer expectations is discussed elsewhere.

object role

— **argv[argc]** shall be a null pointer. 170

Commentary

Some algorithmic styles prefer to decrement a count of the remaining elements. Some prefer to walk a data structure until a null pointer is encountered. The standard supports both styles.

---

171 — If the value of **argc** is greater than zero, the array members **argv[0]** through **argv[argc-1]** inclusive shall contain pointers to strings, which are given implementation-defined values by the host environment prior to program startup.

argv values

Commentary

The storage occupied by the strings has static storage-duration. This wording could be interpreted to imply an ordering between the contents of the host environment and the contents of argv. What the user typed on a command line may not be what gets passed to the program. The host environment is likely to process the arguments first, possibly performing such operations as expanding wildcards and environment variables.

static storage duration

Common Implementations

A white-space character is usually used to delimit program parameters. Many hosts offer some form of quoting mechanism to allow any representable character, in the host environment, to be passed as a value to argv (including character sequences that contain white space). The behavior most often encountered is that the elements of argv, in increasing subscript order, correspond to the white-space delimited character sequences appearing on the command line in left-to-right order.

POSIX defines _POSIX_ARG_MAX, in **<limits.h>**, as "The length of the arguments for one of the exec functions, in bytes, including environment data." and requires that it have a value of at least 4,096. Under MS-DOS the command processor, command.com, reads a value on startup that specifies the maximum amount of the space to use for its environment (where command line information is held).

Coding Guidelines

Most implementations have a limit on the total number of characters, over all the strings, that can be passed via argv. There is a possibility that one of the character sequences in one of the strings pointed to by argv will be truncated.

For GUI-based applications the values passed to argv may be hidden from the user of the application.

Many hosts offer some form of quoting mechanism to allow any typeable character, in that environment, to be passed to a program. If argv is used, a program may need to handle strings containing characters that are not contained in the basic execution character set.

---

172 The intent is to supply to the program information determined prior to program startup from elsewhere in the hosted environment.

main parameters intent

Commentary

The argc/argv mechanism provides a simple-to-use, from both the applications' users' point of view and the developer's, method of passing information (usually frequently changing) to a program. The alternative being to use a file, read by the application, which would need to be created or edited every time options changed. This is not to say that an implementation might not choose to obtain the information by reading from a file (e.g., in a GUI environment where there is no command line).

C++

The C++ Standard does not specify any intent behind its support for this functionality.

Common Implementations

If the program was invoked from the command line, the arguments are usually the sequence of characters appearing after the name of the program on the line containing the program image name. The order of the strings appearing in successive elements of the argv array is the same order as they appeared on the command line and the space character is treated as the delimiter.

In GUI environments more information is likely to be contained in configuration files associated with the program to be executed. Many GUIs implement click, or drag-and-drop, by invoking a program with `argv[1]` denoting the name of the clicked, or dropped-on, file and `argv[2]` denoting the name of the dropped file, or some similar convention.

Coding Guidelines

Making use of `argv` necessarily relies on implementation-defined behavior. Where the information is obtained, and the ordering of strings in `argv`, are just some of the issues that need to be considered.

---

argv lowercase

If the host environment is not capable of supplying strings with letters in both uppercase and lowercase, the implementation shall ensure that the strings are received in lowercase. 173

Commentary

This choice reflects C's origins in Unix environments. Some older host environments convert, by default, all command line character input into a single case. A commonly occurring program argument is the name of a file. Unix filenames tend to be in lowercase and the file system is case-sensitive. Having an application accept non-filename arguments in lowercase has a lower cost than insisting that filenames be in uppercase.

C++

The C++ Standard is silent on this issue.

Common Implementations

In some mainframe environments, the convention is often to use uppercase letters. Although entering lowercase letters is difficult, it has usually proven possible to implement.

Coding Guidelines

Most modern hosts support the use of both upper- and lowercase characters. The extent to which a program needs to take account of those cases where they might not both be available will depend on the information passed via `argv` and the likelihood that the program will need to be ported to such a host.

---

argv program name

— If the value of **`argc`** is greater than zero, the string pointed to by **`argv[0]`** represents the *program name*; 174

Commentary

What is the program name? It is usually thought of as the sequence of characters used to invoke the program, which in turn relates in some way to the program image (perhaps the name of a file).

Common Implementations

Under MS-DOS typing, *abc* on the command line causes the program `abc.exe` to be executed. In most implementations, the value of `argv[0]` in this case is *abc.exe* even though the extension `.exe` may not have appeared on the command line.

Unix based environments use the character sequence typed on the command line as the name of the program. This does not usually include the search path along which the program was found. For symbolic links the name of the symbolic link is usually used, not the name of the file linked to.

Coding Guidelines

Some programs use the name under which they were invoked to modify their behavior, for instance, the GNU file compression program is called `gzip`. Running the same program, having renamed it to `gunzip`, causes it to uncompress files. Running the same program, renamed to `gzcat`, causes it to read its input from `stdin` and send its output to `stdout`. The standard distribution of this software has a single program image and various symbolic links to it, each with different names.

The extent to which the behavior of a program depends on the value of `argv[0]` is outside the scope of these coding guidelines.

`argv[0][0]` shall be the null character if the program name is not available from the host environment. 175

Commentary

Not all host environments can provide information on the name of the currently executing program.

Common Implementations

Most hosts can provide the name of the program via `argv[0]`.

---

176 If the value of **argc** is greater than one, the strings pointed to by **argv[1]** through **argv[argc-1]** represent the *program parameters*.

program parameters

Commentary

This defines the term *program parameters*. It does not require that the value of the parameters come from the command line, although this is the background to the terminology.

Example

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[])
4  {
5  printf("The program parameters were:\n");
6
7  for (int a_index = 0; a_index < argc; a_index++)
8     printf("%s\n", argv[a_index]);
9  }
```

---

177 — The parameters **argc** and **argv** and the strings pointed to by the **argv** array shall be modifiable by the program, and retain their last-stored values between program startup and program termination.

Commentary

Although the strings pointed to by the `argv` array are required to be modifiable, the array itself is not required to be modifiable. Calling `main` recursively (permitted in C90, but not in C99) does not cause the original values to be assigned to those parameters.

C++

The C++ Standard is silent on this issue.

Common Implementations

The addresses of the storage used to hold the `argv` strings may be disjoint from the stack and heap storage areas assigned to a program on startup. Some implementations choose to place the program parameter strings in an area of storage specifically reserved, by the host environment, for programs' parameters.

References

1. HP. *DEC C Language Reference Manual*. Compaq Computer Corporation, aa-rh9na-te edition, July 1999.