

The New C Standard (Excerpted material)

An Economic and Cultural Commentary

Derek M. Jones

derek@knosof.co.uk

3.14

object

object

region of data storage in the execution environment, the contents of which can represent values

Commentary

Objects are created by definitions and calls to the library memory-allocation functions. The region of data storage for objects is a contiguous sequence of bytes. Representation of values within the region of data storage allocated to objects is discussed in great detail later in the standard.

C++

While an *object* is also defined, 1.8p1, to be a region of storage in C++, the term has many other connotations in an object-oriented language.

Other Languages

What C calls objects are often called variables in other languages.

Coding Guidelines

Many developers use the term *variable* to refer to what the C Standard calls an object. The introduction of the term *object-oriented* has meant that few developers use the term *object* in the C sense. Trying to change developer terminology in this case is liable to lead to confusion, and to be of little use (apart from being technically correct).

Example

```

1  #include <stdlib.h>
2
3  extern int glob; /* Declaration of a named object. */
4
5  void f(long param) /* A parameter is also an object. */
6  {
7  typedef short S_17; /* No storage created, not an object. */
8
9  float * p_f = (float *)malloc(sizeof(float)); /* Create an unnamed object. */
10 }

```

The response to DR #042 (question 2) specified that the bytes from which an object is composed need not correspond to any type declared in the program. For instance, a contiguous sequence of elements within an array can be regarded as an object in its own right. Nonoverlapping portions of an array can be regarded as objects in their own right.

```

1  #include <stdlib.h>
2  #include <string.h>
3
4  extern int N;
5
6  void DR_042(void)
7  {
8  void *p = malloc(2*N); /* Allocate an object. */
9
10     {
11     char (*q)[N] = p; /*
12                        * The object pointed to by p may be interpreted
13                        * as having type (char [2][N]) when referenced
14                        * through q.
15                        */
16     memcpy(q[1], q[0], N);
17     }

```

object
contiguous
sequence of bytes
value rep-
resentation
object rep-
resentation

```

18     {
19     char *r = p; /*
20                 * The object pointed to by p may be interpreted as
21                 * having type (char [2*N]) when referenced through r.
22                 */
23     memcpy(r+N, r, N);
24     }
25 }

```

70 NOTE When referenced, an object may be interpreted as having a particular type; see 6.3.2.1.

reference
object

Commentary

A reference that is also an access requires the bits making up an object to be interpreted as a value. This requires that it be interpreted as having a particular type. A reference that does not interpret the contents of an object, for instance an argument to the `memcpy` library function, does not need to interpret it as having a particular type. The issue of the type of an object, when it is referenced, is dealt with in detail elsewhere.

access
particular
type

effective type

C++

The properties of an object are determined when the object is created.

1.8p1

This referenced/creation difference, compared to C, is possible in C++ because it contains the **new** and **delete** operators (as language keywords) for dynamic-storage allocation. The type of the object being created is known at the point of creation, which is not the case when the `malloc` library function is used (one of the reasons for the introduction of the concept of effective type in C99).

effective type

Coding Guidelines

A guideline recommendation dealing with the types that may be used to reference a given object is discussed elsewhere.

?? object
effective type
shall stay the same

Example

```

1  static int glob;
2
3  void f(void)
4  {
5  unsigned char *p_uc = (unsigned char *)&glob;
6
7  glob = 3; /* Object glob interpreted to have type int. */
8
9  *p_uc = 3; /* Same object interpreted as an array of unsigned char. */
10 }

```

References