definitions

|  |  |  |
| --- | --- | --- |
| Foreword |  | Unicode characters and strings (<uchar.h>) (originally specified in ISO/IEC TR 19769:2004) |
| Introduction |  | The Working Group responsible for this standard (WG 14) maintains a site on the World Wide Web at <http://www.open-std.org/JTC1/SC22/WG14/> containing additional information relevant to this standard such as a Rationale for many of the decisions made during its preparation and a log of Defect Reports and Responses. |
| 2. Normative references |  | ISO/IEC 2382-1:1993, Information technology — Vocabulary — Part 1: Fundamental terms. |
| 3.1 | access | <execution-time action> to read or modify the value of an object  NOTE 1 Where only one of these two actions is meant, "read" or "modify" is used.  NOTE 2 "Modify" includes the case where the new value being stored is the same as the previous value.  NOTE 3 Expressions that are not evaluated do not access objects. |
| 3.2 | alignment | requirement that objects of a particular type be located on storage boundaries with addresses that are particular multiples of a byte address |
| 3.3 | argument  actual argument  actual parameter (deprecated) | expression in the comma-separated list bounded by the parentheses in a function call expression, or a sequence of preprocessing tokens in the comma-separated list bounded by the parentheses in a function-like macro invocation |
| 3.4 | behavior | external appearance or action |
| 3.4.1 | implementation-defined behavior | unspecified behavior where each implementation documents how the choice is made  EXAMPLE An example of implementation-defined behavior is the propagation of the high-order bit when a signed integer is shifted right. |
| 3.4.2 | locale-specific behavior | behavior that depends on local conventions of nationality, culture, and language that each implementation documents  EXAMPLE An example of locale-specific behavior is whether the islower function returns true for characters other than the 26 lowercase Latin letters. |
| 3.4.3 | undefined behavior | behavior, upon use of a nonportable or erroneous program construct or of erroneous data, for which this International Standard imposes no requirements  NOTE Possible undefined behavior ranges from ignoring the situation completely with unpredictable results, to behaving during translation or program execution in a documented manner characteristic of the environment (with or without the issuance of a diagnostic message), to terminating a translation or execution (with the issuance of a diagnostic message).  EXAMPLE An example of undefined behavior is the behavior on integer overflow. |
| 3.4.4 | unspecified behavior | use of an unspecified value, or other behavior where this International Standard provides two or more possibilities and imposes no further requirements on which is chosen in any instance  EXAMPLE An example of unspecified behavior is the order in which the arguments to a function are evaluated. |
| 3.5 | bit | unit of data storage in the execution environment large enough to hold an object that may have one of two values  NOTE It need not be possible to express the address of each individual bit of an object. |
| 3.6 | byte | addressable unit of data storage large enough to hold any member of the basic character set of the execution environment  NOTE 1 It is possible to express the address of each individual byte of an object uniquely.  NOTE 2 A byte is composed of a contiguous sequence of bits, the number of which is implementation-defined. The least significant bit is called the low-order bit; the most significant bit is called the high-order bit. |
| 3.7 | character | <abstract> member of a set of elements used for the organization, control, or representation of data |
| 3.7.1 | character | single-byte character  <C> bit representation that fits in a byte |
| 3.7.2 | multibyte character | sequence of one or more bytes representing a member of the extended character set of either the source or the execution environment  NOTE The extended character set is a superset of the basic character set. |
| 3.7.3 | wide character | value representable by an object of type wchar\_t, capable of representing any character in the current locale |
| 3.8 | constraint | restriction, either syntactic or semantic, by which the exposition of language elements is to be interpreted |
| 3.9 | correctly rounded result | representation in the result format that is nearest in value, subject to the current rounding mode, to what the result would be given unlimited range and precision |
| 3.10 | diagnostic message | message belonging to an implementation-defined subset of the implementation's message output |
| 3.11 | forward reference | reference to a later subclause of this International Standard that contains additional information relevant to this subclause |
| 3.12 | implementation | particular set of software, running in a particular translation environment under particular control options, that performs translation of programs for, and supports execution of functions in, a particular execution environment |
| 3.13 | implementation limit | restriction imposed upon programs by the implementation |
| 3.14 | memory location | either an object of scalar type, or a maximal sequence of adjacent bit-fields all having nonzero width  NOTE 1 Tw o threads of execution can update and access separate memory locations without interfering  with each other.  NOTE 2 A bit-field and an adjacent non-bit-field member are in separate memory locations. The same applies to two bit-fields, if one is declared inside a nested structure declaration and the other is not, or if the two are separated by a zero-length bit-field declaration, or if they are separated by a non-bit-field member declaration. It is not safe to concurrently update two non-atomic bit-fields in the same structure if all members declared between them are also (non-zero-length) bit-fields, no matter what the sizes of those intervening bit-fields happen to be.  EXAMPLE A structure declared as   |  | | --- | | struct {  char a;  int b:5, c:11, :0, d:8;  struct { int ee:8; } e;  } |   contains four separate memory locations: The member a, and bit-fields d and e.ee are each separate memory locations, and can be modified concurrently without interfering with each other. The bit-fields b and c together constitute the fourth memory location. The bit-fields b and c cannot be concurrently modified, but b and a, for example, can be. |
| 3.15 | object | region of data storage in the execution environment, the contents of which can represent values  NOTE When referenced, an object may be interpreted as having a particular type; see 6.3.2.1. |
| 3.16 | parameter  formal parameter  formal argument (deprecated) | object declared as part of a function declaration or definition that acquires a value on entry to the function, or an identifier from the comma-separated list bounded by the parentheses immediately following the macro name in a function-like macro definition |
| 3.17 | recommended practice | specification that is strongly recommended as being in keeping with the intent of the standard, but that may be impractical for some implementations |
| 3.18 | runtime-constraint | requirement on a program when calling a library function  NOTE 1 Despite the similar terms, a runtime-constraint is not a kind of constraint as defined by 3.8, and need not be diagnosed at translation time.  NOTE 2 Implementations that support the extensions in annex K are required to verify that the runtime-  constraints for a library function are not violated by the program; see K.3.1.4. |
| 3.19 | value | precise meaning of the contents of an object when interpreted as having a specific type |
| 3.19.1 | implementation-defined value | unspecified value where each implementation documents how the choice is made |
| 3.19.2 | indeterminate value | either an unspecified value or a trap representation |
| 3.19.3 | unspecified value | valid value of the relevant type where this International Standard imposes no requirements on which value is chosen in any instance  NOTE An unspecified value cannot be a trap representation. |
| 3.19.4 | trap representation | an object representation that need not represent a value of the object type |
| 3.19.5 | perform a trap | interrupt execution of the program such that no further operations are performed  NOTE In this International Standard, when the word "trap" is not immediately followed by "representation", this is the intended usage. 2) |
| 3.20 | ... | ceiling of x: the least integer greater than or equal to x  EXAMPLE ... |
| 3.21 | ... | floor of x: the greatest integer less than or equal to x  EXAMPLE ... |
| 4. Conformance | undefined behavior | If a "shall" or "shall not" requirement that appears outside of a constraint or runtime-constraint is violated, the behavior is undefined. Undefined behavior is otherwise indicated in this International Standard by the words "undefined behavior" or by the omission of any explicit definition of behavior. There is no difference in emphasis among these three; they all describe "behavior that is undefined". |
| 4. Conformance | strictly conforming program | A strictly conforming program shall use only those features of the language and library specified in this International Standard. 3) It shall not produce output dependent on any unspecified, undefined, or implementation-defined behavior, and shall not exceed any minimum implementation limit. |
| 4. Conformance | conforming implementation | The two forms of conforming implementation are hosted and freestanding. A conforming hosted implementation shall accept any strictly conforming program. A conforming freestanding implementation shall accept any strictly conforming program in which the use of the features specified in the library clause (clause 7) is confined to the contents of the standard headers <float.h>, <iso646.h>, <limits.h>, <stdalign.h>, <stdarg.h>, <stdbool.h>, <stddef.h>, <stdint.h>, and <stdnoreturn.h>. A conforming implementation may have extensions (including additional library functions), provided they do not alter the behavior of any strictly conforming program. 4)  4) This implies that a conforming implementation reserves no identifiers other than those explicitly reserved in this International Standard. |
| 4. Conformance | conforming program | A conforming program is one that is acceptable to a conforming implementation. 5)  5) Strictly conforming programs are intended to be maximally portable among conforming implementations. Conforming programs may depend upon nonportable features of a conforming implementation. |
| 5. Environment |  | An implementation translates C source files and executes C programs in two data-processing-system environments, which will be called the translation environment and the execution environment in this International Standard. |
| 5.1.1.1 Program structure | source files  preprocessing files | The text of the program is kept in units called source files, (or preprocessing files) in this International Standard. |
| 5.1.1.1 Program structure | preprocessing translation unit | A source file together with all the headers and source files included via the preprocessing directive #include is known as a preprocessing translation unit. |
| 5.1.1.1 Program structure | translation unit | After preprocessing, a preprocessing translation unit is called a translation unit. |
| 5.1.1.1 Program structure | translated translation unit |  |
| 5.1.1.1 Program structure | new-line character |  |
| 5.1.1.1 Program structure | end-of-line indicator |  |
| 5.1.1.2 Translation phases | physical source line  logical source line | Each instance of a backslash character (\) immediately followed by a new-line character is deleted, splicing physical source lines to form logical source lines. |
| 5.1.1.2 Translation phases | program image |  |
| 5.1.2 Execution environments |  | Two execution environments are defined: freestanding and hosted. |
| 5.1.2 Execution environments | program startup | In both cases, program startup occurs when a designated C function is called by the execution  environment. |
| 5.1.2 Execution environments | initialize | All objects with static storage duration shall be initialized (set to their initial values) before program startup. |
| 5.1.2 Execution environments | program termination | Program termination returns control to the execution environment. |
| 5.1.2.1 Freestanding environment | freestanding environment | In a freestanding environment (in which C program execution may take place without any benefit of an operating system), the name and type of the function called at program startup are implementation-defined. |
| 5.1.2.2.1 Program startup | main | The function called at program startup is named main. |
| 5.1.2.2.1 Program startup | program name | If the value of argc is greater than zero, the string pointed to by argv[0] represents the program name; argv[0][0] shall be the null character if the program name is not available from the host environment. |
| 5.1.2.2.1 Program startup | program parameters | If the value of argc is greater than one, the strings pointed to by argv[1] through argv[argc-1] represent the program parameters. |
| 5.1.2.2.2 Program execution | （hosted environment） | In a hosted environment, a program may use all the functions, macros, type definitions, and objects described in the library clause (clause 7). |
| 5.1.2.3 Program execution | side effects | Accessing a volatile object, modifying an object, modifying a file, or calling a function that does any of those operations are all side effects, 12) which are changes in the state of the execution environment. |
| 5.1.2.3 Program execution | evaluation | Evaluation of an expression in general includes both value computations and initiation of side effects. Value computation for an lvalue expression includes determining the identity of the designated object. |
|  | sequenced before  sequenced after |  |
|  | unsequenced |  |
|  | indeterminately sequenced |  |
|  | sequence point |  |
| 5.1.2.3 Program execution | observable behavior | The least requirements on a conforming implementation are:  — Accesses to volatile objects are evaluated strictly according to the rules of the abstract machine.  — At program termination, all data written into files shall be identical to the result that execution of the program according to the abstract semantics would have produced.  — The input and output dynamics of interactive devices shall take place as specified in 7.21.3. The intent of these requirements is that unbuffered or line-buffered output appear as soon as possible, to ensure that prompting messages actually appear prior to a program waiting for input.  This is the observable behavior of the program. |
|  | implicit spilling |  |
|  | explicit store and load |  |
|  | roundoff error |  |
| 5.1.2.4 Multi-threaded executions and data races | thread of execution (or thread)  The execution of each thread  The execution of the entire program | Under a hosted implementation, a program can have more than one thread of execution (or thread) running concurrently. The execution of each thread proceeds as defined by the remainder of this standard. The execution of the entire program consists of an execution of all of its threads. 14)  14) The execution can usually be viewed as an interleaving of all of the threads. However, some kinds of atomic operations, for example, allow executions inconsistent with a simple interleaving as described below. |
|  | conflict | Two expression evaluations conflict if one of them modifies a memory location and the other one reads or modifies the same memory location. |
|  | atomic operations |  |
|  | synchronization  operation |  |
|  | acquire operation  release  operation  consume operation |  |
|  | fence |  |
|  | acquire fence  release fence |  |
|  | relaxed atomic operation |  |
|  | read-modify-write operation |  |
|  | ... |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

implementation-defined behaviours