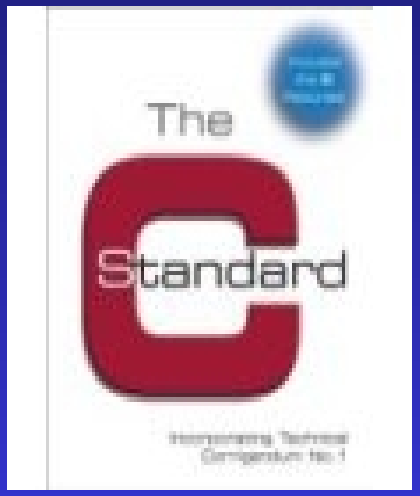
Standard

chapter and verse

- http://www.open-std.org/jtc1/sc22/wg14



Joint Technical Committee 1

Sub Committee 22

Working Group 14

John Wiley: ISBN 0470845732

- http://www.knosoft.co.uk/cbook/cbook.html
  - commentary on every sentence in the standard+

- Peter Norvig advises
  - get involved in a language standardization
  - get off the language standardization asap :-)

Clauses

3. Terms, definitions, and symbols
4. Conformance
5. Environment
5.1.1.3 Diagnostics
5.1.2.3 Program execution
6. Language
6.2 Concepts
6.2.5 Types
6.2.6 Representations of types
6.3 Conversions
6.3.2.1 Lvalues, arrays, and function designators
6.5 Expressions
6.7 Declarations
6.8 Statements and blocks

These are the main clauses we will be looking at

spirit of C

- trust the programmer
  - let them do what needs to be done
  - the programmer is in charge not the compiler
- keep the language small and simple
  - provide only one way to do an operation
  - new inventions are not entertained
- make it fast, even if its not portable
  - target efficient code generation
  - int promotion rules
  - sequence points
- rich expressions
  - lots of operators
  - expressions combine into larger expressions

implementation

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 executions of functions in, a particular execution environment.

5. Environment
para 1 - 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...

Implementation means compiler/translator

behavior

3.4 behavior
external appearance or action

3.4.1 implementation-defined behavior
unspecified behavior where each implementation documents how the choice is made.

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 requirement.

3.4.4 unspecified behavior
use of an unspecified value, or other behavior where this International Standard produces two or more possibilities and impose no further requirements on which is chosen in any instance.

question

- behaviour in Java/C# is completely defined
- a lot of behaviour in C is not - why not?

> *3.4 behavior*
> *...*

> *3.4.1 implementation-defined behavior*
> *...*

> *3.4.3 undefined behavior*
> *...*

> *3.4.4 unspecified behavior*
> *...*

diagnostics

3.8 constraint
restriction, either syntactic or semantic, by which the exposition of language elements is to be interpreted.

3.10 diagnostic message
message belonging to an implementation-defined subset of the implementation's message output.

5.1.1.3 Diagnostics
para 1 - A conforming implementation shall produce at least one diagnostic message ... if a ... translation unit contains a violation of any syntax rule or constraint ...
Diagnostic messages need not be produced in other circumstances.

- **an implementation is required to produce a diagnostic message for a *syntax* violation***

> ***6.5.16 Assignment operators***
> ***Syntax***
> ***assignment-operator:***
> = *= /= %= += -= <<= >>= &= ^= |=

syntax.c

```
int x = 0;
x := 42;
```

```
>gcc syntax.c
  → error: expected expression before '=' token
```

*if it is the first violation

- **an implementation is required to produce a diagnostic message for a *constraint* violation***

> ***6.5.16 Assignment operators***
> ***Constraints***
> ***As assignment operator shall have a modifiable lvalue as its left operand.***

constraints.c

```
const int x = 0;
x = 42;
```

```
>gcc constraints.c
  → error: assignment of read-only variable 'x'
```

*if it is the first violation

- **an implementation is *not* required to produce a diagnostic message for any other violation!**

> ***5.1.2.3 Program execution***
> ***para 2 – At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place.***

semantics.c

```
int x = 0;
x = x++;
```

```
>gcc semantics.c
  → ...no diagnostic...
```

```
>gcc –Wall semantics.c
  → warning: operation on 'x' may be undefined
```

conformance

4. Conformance
para 1 - ... "shall" is to be interpreted as a requirement on an implementation or on a program; conversely, "shall not" is to be interpreted as a prohibition.

para 2 - If a "shall" or "shall not" requirement that appears outside of a 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; they all describe "behavior that is undefined"

The word shall appears 454 times outside of a Constraint clause.
Annex J.2 lists 190 undefined behaviours.

conformance

4. Conformance
para 5 - A strictly conforming program shall use only those features of the language and library specified in this International Standard.
It shall not produce output dependent on any unspecified, undefined, or implementation-defined behavior and shall not exceed any minimum implementation limit.

para 6 - A conforming ... implementation shall accept any strictly conforming program.

para 7 - A conforming program is one that is acceptable to a conforming implementation.

The standard never writes about "correct" programs.
The only proper terms are conformance/conforming.

3. Terms, definitions, and symbols

3.14 object

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

What C calls objects, other languages call variables.

All object's representation is held in a contiguous sequence of bytes.

An object is addressable.

objects

3. Terms, definitions, and symbols

3.14 object

NOTE: when referenced, an object may be interpreted as having a particular type.

A reference that does not interpret the contents of an object, for example as an argument to memcpy, does not need to interpret it as having a particular type.

Objects have storage duration but no linkage.

Identifiers have linkage but no storage duration.

Constants have only a value.

objects

3. Terms, definitions, and symbols

3.6 byte

Addressable unit of data storage large enough to hold any member of the basic character set of the execution environment

Each byte is at least 8 bits wide.

A char whether signed or unsigned occupies exactly one byte.

An implementation cannot hide the internal bits of an object.

Why not?

bytes

values

- 3. Terms, definitions, and symbols
  - 3.17 value

Precise meaning of the contents of an object when interpreted as having a specific type.

A literal also has a value. Its type is determined by both the lexical form of the token and its numeric value.

- # 3. Terms, definitions, and symbols
  - ## 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 in the same as the previous value.
NOTE 3: Expressions that are not evaluated do not access objects.

access

What use is an expression that is not evaluated?

- 6. Language
  - 6.2 Concepts
    - 6.2.5 Types

para 1 - The meaning of a value stored in an object or returned by a function is determined by the type of the expression used to access it.

para 1 – Types are partitioned into
- *object types (types that fully describe objects)*
- *function types (types that describe functions), and*
- *incomplete types (types that describe objects but lack information needed to determine their sizes).*

C has a relatively weak type system. C++ added much greater support for "typing".

types

expressions

- 6. Language
  - 6.5 Expressions

> *para 1 - An expression is a sequence of operators and operands that ...*
> - *specifies computation of a value, or*
> - *that designates an object or a function, or*
> - *that generates side effects, or*
> - *that performs a combination thereof.*

- rich expressions
  - lots of operators
  - expressions combine into larger expressions

side effects

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 2 –
- *Accessing a volatile object,*
- *modifying an object,*
- *modifying a file,*
- *or calling a function that does any of those operations*

are all side effects which are changes in the state of the execution environment.

sequence points

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 2 – At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place.

Why do so few expressions cause a sequence point?

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 3 – In the abstract machine, all expressions are evaluated as specified by the semantics.
An actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no needed side effects are produced (including any caused by calling a function or accessing a volatile object).

Why does the as-if rule exist?

as-if

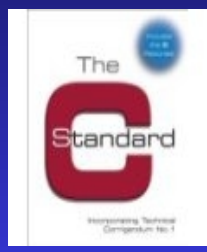
contents

- Spirit of C
- Main clauses
- Behaviour
- Diagnostics
- Conformance
- Key terms
- Types
- Expressions
- Program execution

Standard

chapter and verse

- http://www.open-std.org/jtc1/sc22/wg14



Joint Technical Committee 1

Sub Committee 22

Working Group 14

John Wiley: ISBN 0470845732

- http://www.knosoft.co.uk/cbook/cbook.html
  - commentary on every sentence in the standard+
- Peter Norvig advises
  - get involved in a language standardization
  - get off the language standardization asap :-)

Clauses

3. Terms, definitions, and symbols
4. Conformance
5. Environment
5.1.1.3 Diagnostics
5.1.2.3 Program execution
6. Language
6.2 Concepts
6.2.5 Types
6.2.6 Representations of types
6.3 Conversions
6.3.2.1 Lvalues, arrays, and function designators
6.5 Expressions
6.7 Declarations
6.8 Statements and blocks

These are the main clauses we will be looking at

spirit of C

- trust the programmer
  - let them do what needs to be done
  - the programmer is in charge not the compiler
- keep the language small and simple
  - provide only one way to do an operation
  - new inventions are not entertained
- make it fast, even if its not portable
  - target efficient code generation
  - int promotion rules
  - sequence points
- rich expressions
  - lots of operators
  - expressions combine into larger expressions

implementation

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 executions of functions in, a particular execution environment.

5. Environment
para 1 - 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...

Implementation means compiler/translator

For 3.4.3 undefined behavior, the c90 standard said

"Behavior, upon use of a nonportable or errnoeous program construct or of erroneous data, or of indeterminately valued objects, for which this International Standard imposes no requirements."

The reason for dropping the word "or of indeterminately valued objects" is that it is always possible to read the value of an object whose type is unsigned char. Objects of type unsigned char never support trap representations.

question

- behaviour in Java/C# is completely defined
- a lot of behaviour in C is not - why not?

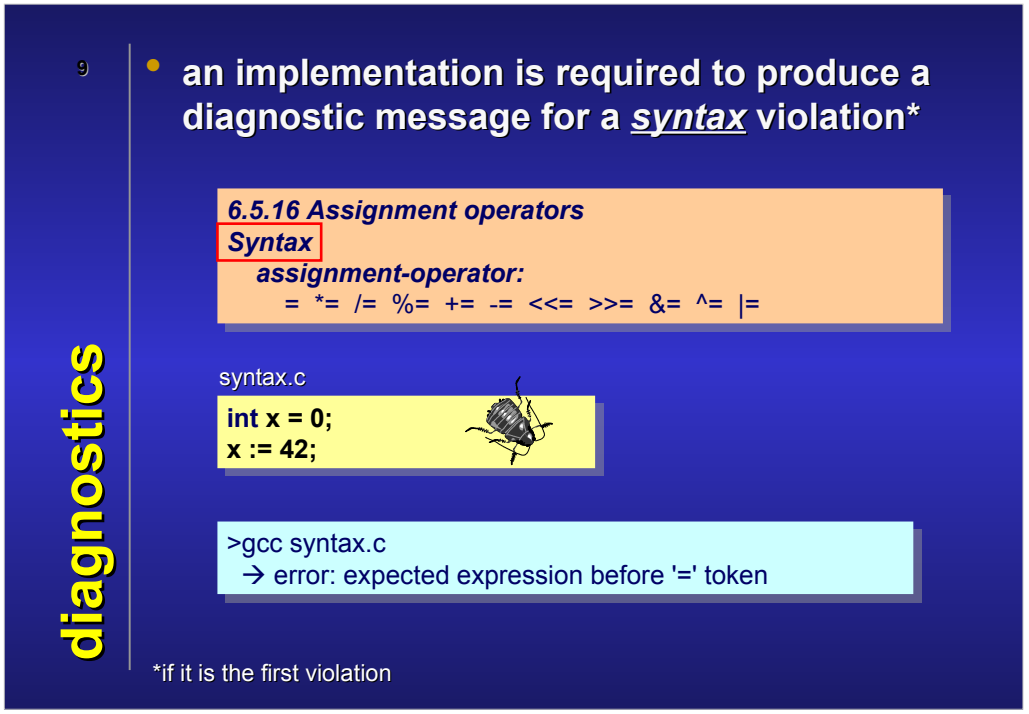
> *3.4 behavior*
> *...*
>
> *3.4.1 implementation-defined behavior*
> *...*
>
> *3.4.3 undefined behavior*
> *...*
>
> *3.4.4 unspecified behavior*
> *...*

diagnostics

3.8 constraint
restriction, either syntactic or semantic, by which the exposition of language elements is to be interpreted.

3.10 diagnostic message
message belonging to an implementation-defined subset of the implementation's message output.

5.1.1.3 Diagnostics
para 1 - A conforming implementation shall produce at least one diagnostic message ... if a ... translation unit contains a violation of any syntax rule or constraint ...
Diagnostic messages need not be produced in other circumstances.



If there is more than one syntax error in a translation unit the implementation is only required to produce a single diagnostic. If a programmer corrects the first syntax error the effect is simply to make the second syntax error the new first syntax error and thus another diagnostic is produced.

This is best understood in reverse: if there are no syntax errors (or constraint violations) the implementation is not required to produce a diagnostic.

diagnostics

- an implementation is required to produce a diagnostic message for a ***constraint*** violation*

> ***6.5.16 Assignment operators***
> ***Constraints***
> ***As assignment operator shall have a modifiable lvalue as its left operand.***

constraints.c

```
const int x = 0;
x = 42;
```

```
>gcc constraints.c
  → error: assignment of read-only variable 'x'
```

*if it is the first violation

11

diagnostics

- **an implementation is *not* required to produce a diagnostic message for any other violation!**

> ***5.1.2.3 Program execution***
> ***para 2 – At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place.***

semantics.c

```
int x = 0;
x = x++;
```

```
>gcc semantics.c
  → ...no diagnostic...
```

```
>gcc –Wall semantics.c
  → warning: operation on 'x' may be undefined
```

Hopefully the message is clear: because of the nature of C it is imperative to get as much help from the implementation as possible. The more diagnostics the better!

conformance

4. Conformance
para 1 - ... "shall" is to be interpreted as a requirement on an implementation or on a program; conversely, "shall not" is to be interpreted as a prohibition.

para 2 - If a "shall" or "shall not" requirement that appears outside of a 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; they all describe "behavior that is undefined"

The word shall appears 454 times outside of a Constraint clause.
Annex J.2 lists 190 undefined behaviours.

conformance

4. Conformance
para 5 - A strictly conforming program shall use only those features of the language and library specified in this International Standard.
It shall not produce output dependent on any unspecified, undefined, or implementation-defined behavior and shall not exceed any minimum implementation limit.

para 6 - A conforming ... implementation shall accept any strictly conforming program.

para 7 - A conforming program is one that is acceptable to a conforming implementation.

The standard never writes about "correct" programs.
The only proper terms are conformance/conforming.

objects

- 3. Terms, definitions, and symbols
  - 3.14 object

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

What C calls objects, other languages call variables.

All object's representation is held in a contiguous sequence of bytes.

An object is addressable.

objects

- 3. Terms, definitions, and symbols
  - 3.14 object

NOTE: when referenced, an object may be interpreted as having a particular type.

A reference that does not interpret the contents of an object, for example as an argument to memcpy, does not need to interpret it as having a particular type.

Objects have storage duration but no linkage.

Identifiers have linkage but no storage duration.

Constants have only a value.

bytes

- 3. Terms, definitions, and symbols
  - 3.6 byte

Addressable unit of data storage large enough to hold any member of the basic character set of the execution environment

Each byte is at least 8 bits wide.

A char whether signed or unsigned occupies exactly one byte.

An implementation cannot hide the internal bits of an object.

Why not?

values

- 3. Terms, definitions, and symbols
  - 3.17 value

Precise meaning of the contents of an object when interpreted as having a specific type.

A literal also has a value. Its type is determined by both the lexical form of the token and its numeric value.

access

- 3. Terms, definitions, and symbols
  - 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 in the same as the previous value.
NOTE 3: Expressions that are not evaluated do not access objects.

What use is an expression that is not evaluated?

types

- 6. Language
  - 6.2 Concepts
    - 6.2.5 Types

para 1 - The meaning of a value stored in an object or returned by a function is determined by the type of the expression used to access it.

para 1 – Types are partitioned into
- *object types (types that fully describe objects)*
- *function types (types that describe functions), and*
- *incomplete types (types that describe objects but lack information needed to determine their sizes).*

C has a relatively weak type system. C++ added much greater support for "typing".

expressions

- 6. Language
  - 6.5 Expressions

> *para 1 - An expression is a sequence of operators and operands that ...*
> *• specifies computation of a value, or*
> *• that designates an object or a function, or*
> *• that generates side effects, or*
> *• that performs a combination thereof.*

- rich expressions
  - lots of operators
  - expressions combine into larger expressions

side effects

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 2 –
• Accessing a volatile object,
• modifying an object,
• modifying a file,
• or calling a function that does any of those operations are all side effects which are changes in the state of the execution environment.

sequence points

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 2 – At certain specified points in the execution sequence called sequence points, all side effects of previous evaluations shall be complete and no side effects of subsequent evaluations shall have taken place.

Why do so few expressions cause a sequence point?

23

- 5. Environment
  - 5.1 Conceptual models
    - 5.1.2 Execution environment
      - 5.1.2.3 Program execution

para 3 – In the abstract machine, all expressions are evaluated as specified by the semantics.
An actual implementation need not evaluate part of an expression if it can deduce that its value is not used and that no needed side effects are produced (including any caused by calling a function or accessing a volatile object).

Why does the as-if rule exist?

as-if

The abstract machine closely follows the developers intuition on what they think their programs do and what they typically do when there are being debugged with no optimization flags being set.

The as-if rule provides maximum leeway to compiler writers to squeeze speed out of the program. Together with the sequence point rules it creates ample opportunity for confusion when debugging a program with optimizations turned on.

contents

- Spirit of C
- Main clauses
- Behaviour
- Diagnostics
- Conformance
- Key terms
- Types
- Expressions
- Program execution