Built-In Functionality

Before you Write C code, consider what you not have to write since it's built-in.

The C Standard Library

- Set of well-specified built-in functionality
- C language implementation-independent
- First specified with ANSI C
- Contains most fundamental macros, types, and functions
- Particular functionality is added to a program by
  1. including particular C standard libraries header files (see later)
     - Declaration of functions
  2. linking function symbols referring to standard library functions
     - Definition is provided by the standard library implementation (i.e., C runtime)
     - Often implemented by mixture of assembler and C itself

Standard Library Modules (1)

```
#include <stdio.h>
```

Input and Output

- generic file operations for byte characters
- I/O streams associated with an external physical device (files, printer,...)
  - Standard input
  - Standard (error) output
- File access functions
  - opening & closing, aliasing, synchronization, buffer operations
- Direct input and output
  - reading from and writing to files
- Unformatted and formatted input and output
  - character/string reading from input stream, writing to output streams
- File positioning, error handling, file operations (remove, rename,...)

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (2)

```
#include <assert.h>
```

Diagnostics

- runtime assertions (user-specified condition to abort program)
  - disable for release builds
  - *example*: check whether argument is null or not
- compile-time assertions (user-specified condition to stop compilation)
  - *example*: check whether implementation-defined type size in in certain boundaries

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (3)

```
#include <complex.h>
```

C99 or later

Complex Number Arithmetics

- complex number math
  - arithmetic operators for real, complex, and imaginary types in any combination
- manipulation functions
  - construction, computations of real, imaginary, magnitude, phase, ...
- exponential functions
  - complex base-e exponential and natural logarithm
- power functions
  - complex power and square root
- trigonometric functions
  - complex sine, cosine, tangent, arc sine, arc cosine, and arc tangent
- hyperbolic functions
  - hyperbolic sine, hyperbolic cosine, hyperbolic tangent, ...

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (4)

```
#include <ctype.h>
```

Null-Terminated Byte Strings Tests

- character classification functions
  - check for alphanumeric, alphabetic, lower/uppercase, digit, hexadecimal, control/ graphical character, spaces/blanks, printing & punctuation character
- character manipulation functions
  - conversion to lowercase and uppercase
- numeric formats conversion
  - byte string to a floating-point, integer, unsigned integer, and floating point
- string manipulation and examination
  - strings/substrings copying & concatenation, localization for string comparing
  - lengths, comparing (sub)strings, first/last occurrence search, string tokenizing
- character array manipulation
  - searching of elements in array, comparing byte arrays, copying & moving (sub) arrays
- printing error codes as human readable (string) messages

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (5)

```
#include <errno.h>
```

Error Reporting

- POSIX-compatible
  - thread-local error numbers
  - error conditions
- boundary checking

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (6)

```
#include <fenv.h>
```

C99 or later

Floating Point Environment

- thread-local set of status flags and control modes after floating-point operations
- status flags indicate abnormal results or auxiliary information
- native support only by few compilers
- functions
  - flag clearing, getting certain flag, exceptions raising (division by zero, inexact, invalid, overflow, underflow), copying flags, getting/setting round directions (downward, towards zero, nearest, upwards), saving/restoring current states,

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (7)

```
#include <float.h>
```

```
#include <limits.h>
```

Implementation-defined Limits of Float and Basic Types

- floating-point types
  - decimal digits, min, max, epsilon, mantissa-specific information, number of digits guaranteed preserved in text, . . .
- basic types
  - library types limits
    - pointer arithmetic, array indexing and loop counting types, atomic integers,. . .
  - integer types limits
    - number bits per byte, max number of bytes in multibyte character
    - min/max value of (signed/unsigned) character, short, int, long, and long long

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (8)

`#include <iso646.h>`

C95 or later

Alternatives to Operator Spellings

C operators and punctuators require `{` , `}` , `[` , `]` , `#` , `\` , `^` , `|` , `~` characters that might not be available in a particular character encoding (e.g., DIN 66003)

Alternatives to those characters:

- macros
  - text substitution, e.g., `and` for `&&` , `and_eq` for `&=` ,...
- digraphs
  - substitute two characters by one, e.g., `<%` for `{` , `<:` for `[` , ...
- trigraphs
  - substitute three characters by one, e.g., `??<` for `{` , `??(` for `[` ,...

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (9)

```
#include <locale.h>
```

Localization Support

Localization is adapting a software to different regions, and different languages.
Adaption includes technical differences (e.g., character encoding).

Localization affects

- character classification and string comparison
- formatting of numbers, currency, and date/time
- behavior of stream I/O, regular expression
- ...

The standard library provides function to individually get and set localization incl. formatting details .

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (10)

```
#include <math.h>
```

Common Mathematical Functions

- basic functions
  - absolute value, quotient and remainder, division, min/max, differences for ints/floats
- exponential and power functions
  - base-2, base-10, and base-e exponential, base-2, base-10m and natural logarithm
  - power, square root, cubic root, hypotenuse calculation
- trigonometric and hyperbolic functions
  - complex sine, cosine, tangent, arc sine, arc cosine, and arc tangent
  - hyperbolic sine, hyperbolic cosine, hyperbolic tangent, ...
- error and gamma functions
  - statistical error, complementary, gamma, and natural logarithm of gamma function
- nearest integer floating-point functions
  - ceil, floor, trunc, rounding with several modes
- floating-point manipulation, float classification and comparison, macros, constants,...

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (11)

```
#include <setjmp.h>
```

Non-local Jumps

Unconditional control flow jump (c.f. `goto` *later*), specialized for jumps between functions.

`jmp_buf` environment and context

- buffer for restoring execution environment, and for invocation of that environment

`setjmp` non-local jumps

- save current context in a buffer used to restore execution context later on

`longjmp` context execution

- loading of execution environment, and transferring control flow to jump target

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (12)

```
#include <signal.h>
```

Signal Handling

Signals is POSIX-compliant form of inter-process communication in terms of asynchronous notifications to processes (or threads) on the happening of an event, such as

- program termination (`SIGABRT`), interruption (`SIGINT`), or killing (`SIGKILL`)
- elapsing of particular timers (`SIGALRM`)
- job control events, e.g., pausing (`SIGSTOP`) or continuing (`SIGCONT`)
- errors, e.g., bus error (`SIGBUS`), or division-by-zero (`SIGFPE`)
- ...

Functions and macros for signal handling in C programs

- setting signal handler for particular signals
- invocation of signals for the current program

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (13)

```
#include <stdalign.h>
```

C11 or later

Alignment Requirements and Alignment Information

Alignment requirement of a type represents the number of bytes (power of two) between successive addresses at which objects of this type can be allocated.

Support of modification and querying of per-type alignment requirements, e.g., of members in structure and unions.

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (14)

```
#include <stdarg.h>
```

Variable Number of Arguments for Functions

Support for functions taking a variable number of arguments (aka variadic functions), e.g., `printf`

declaration: `int foo(int argument, ...);`
usage: `foo(42, 23, „Hello“, „World“);`

Support in function definition enabled by specialized macros to

- enable variadic arguments
- access next variadic argument
- copying variadic arguments
- ends traversal of variadic argument list

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (15)

```
#include <stdatomic.h>
```

C11 or later

Atomic Types and Operations

Requirements and semantics of multi-threaded programs to enable portable multi-threaded programs efficiently manipulating objects in parallel without data races.

- memory order constraints
  - ordering of non-atomic memory accesses around an atomic operation
    - sequentially-consistent ordering, relaxed ordering, release-consume ordering,...
- atomic functions
  - atomic write, read and (conditional) swap operations, atomic test and set and clear, check for lock-freeness, atomic addition, subtraction, logical OR, logical exclusive OR, logical AND, synchronization primitives
- atomic types
  - atomic types for core languages types, guaranteed lock-free atomic flag

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (16)

```
#include <stdbool.h>
```

C99 or later

Boolean Type Support

Older C language specifications don't support dedicated boolean types (i.e., int was used)

Since C99, a boolean type `bool` (a macro expanded to built-in type `_Bool`) can be used

- application of standard logical operators in any combination
- provisioning of two macro constants `true` (integer `1`) and `false` (integer `0`)

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (...)

Format Conversion of Integer Types	`#include <inttypes.h>`
Common Macro Definitions	`#include <stddef.h>`
Fixed Width Integer Types	`#include <stdint.h>`
Memory management, Program Utilities, String Conversions, & Random Numbers	`#include <stdlib.h>`
Annotation for Non-Returning Functions	`#include <stdnoreturn.h>`
Null-Terminated Byte String Handling	`#include <string.h>`
Type-Generic Common Mathematical Functions	`#include <tgmath.h>`

C99 or later

C11 or later

C99 or later

taken from: http://en.cppreference.com/w/c/header

Standard Library Modules (...)

Thread Support	`#include <threads.h>`
Date and Time Support	`#include <time.h>`
Unicode UTF-16 and UTF-32 Characters	`#include <uchar.h>`
Extended Multibyte and Wide Character	`#include <wchar.h>`
Null-terminated Wide Strings	`#include <wctype.h>`

C11 or later

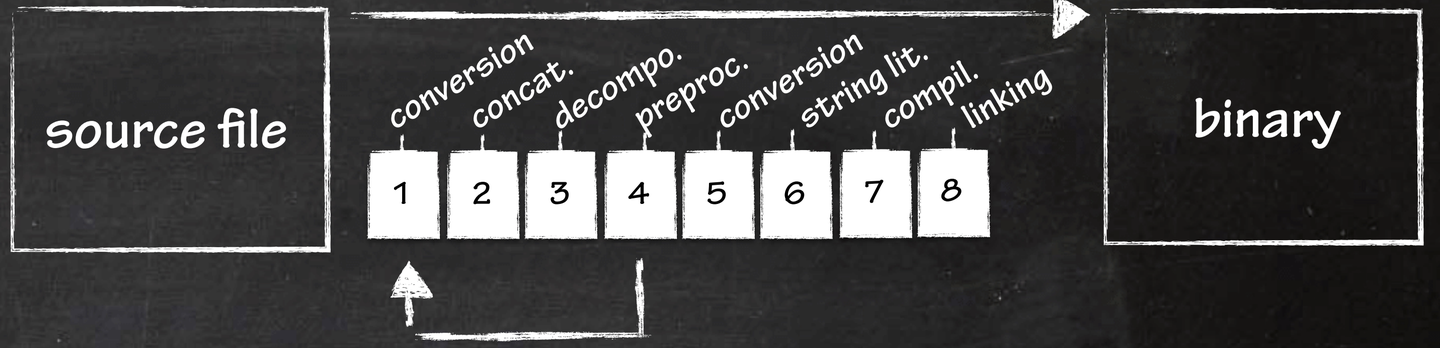
C11 or later

C95 or later

C95 or later

taken from: http://en.cppreference.com/w/c/header

Translation (1)



Phase 1

Source Character Set Conversion. Conversion of the source file text file characters into text containing only characters from the source character set:

- 5 types of whitespace characters
  - space, horizontal tab, vertical tab, form feed, new-line
- 10 digit characters (0 to 9)
- 52 letters ('a' to 'z', 'A' to 'Z')
- 29 punctuation characters (_ { } [ ] # ( ) < > % : ; . ? * + - / ^ & | ~ ! = , \ " ')
- Trigraphs are replaced by single-character representations

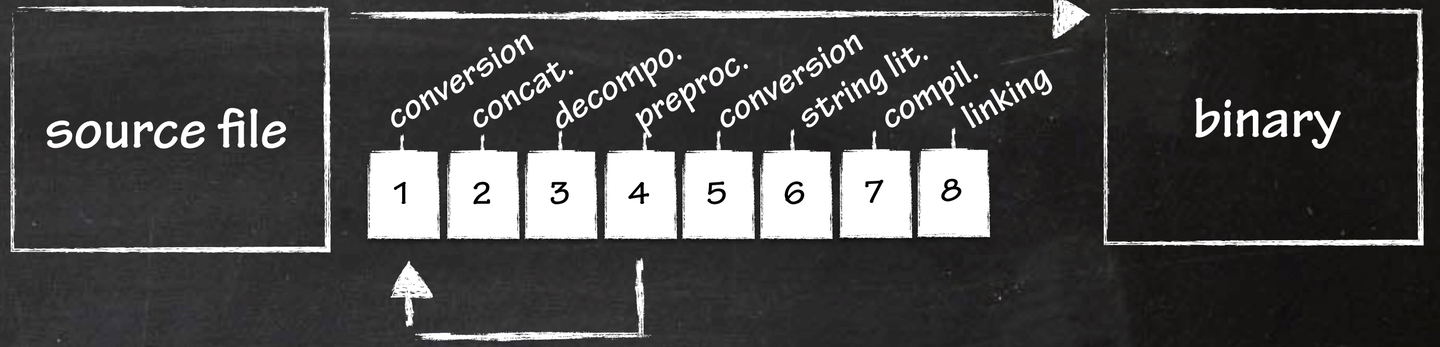
Phase 2

Concatenation. Replacement of new line characters, i.e., concatenate physical lines into one logical line

- does not apply to empty lines
  - long logical lines

based on: http://en.cppreference.com/w/c/language/translation_phases

Translation (2)



Phase 3

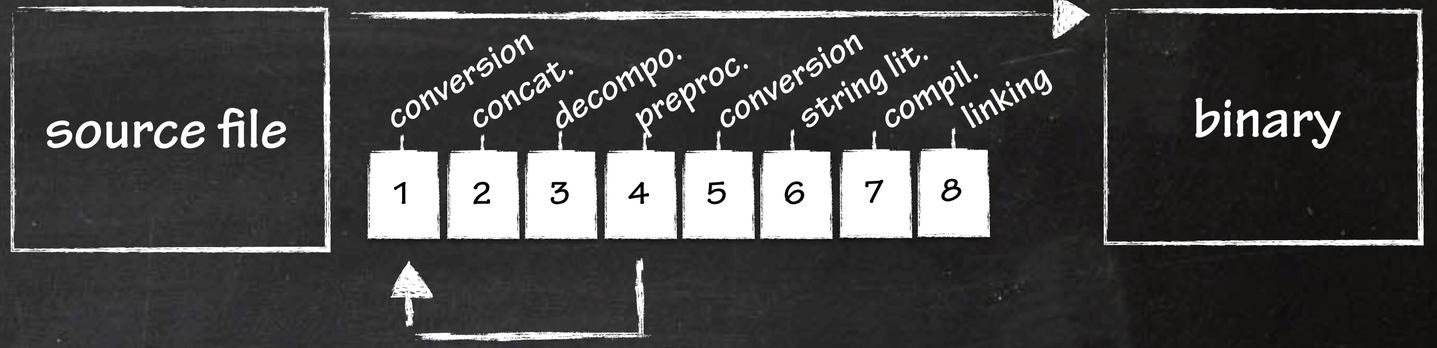
Decomposition into comments, whitespace sequences, and preprocessing tokens:

- header names: <stdio.h> or "file.h"
- identifiers and numbers
- character constants and string literals
- operators
  - ! % ^ & * - + = ~ | . < > / ? : , [ ] ( ) #
  - ++ -- -> << >> <= >= == != *= /=
  - %= += -= <<= >>= &= ^= |= ## && ||
- punctuators
  - < > header name
  - [ ] array delimiter
  - { } initializer list, or function body, or compound statement
  - ( ) function parameter list delimiter;
  - * pointer declaration
  - , list separator
  - : statement label
  - = declaration initializer
  - ; end statement
  - ... variable-length argument list
  - # preprocessor directive
  - ' ' character constant
  - " " string literal or header name

based on: http://en.cppreference.com/w/c/language/translation_phases

based on: https://www.cs.auckland.ac.nz/references/unix/digital/AQTLTBTE/DOCU_008.HTM

Translation (3)



Phase 3

- replacement of comments by one whitespace character
- newlines are untouched

Phase 4

Preprocessing

- sources are preprocessed by the C preprocessor (see later)
- recursively apply phase 1 to 4 for all includes files (#include directive)
  - ends up in complete preprocessed file

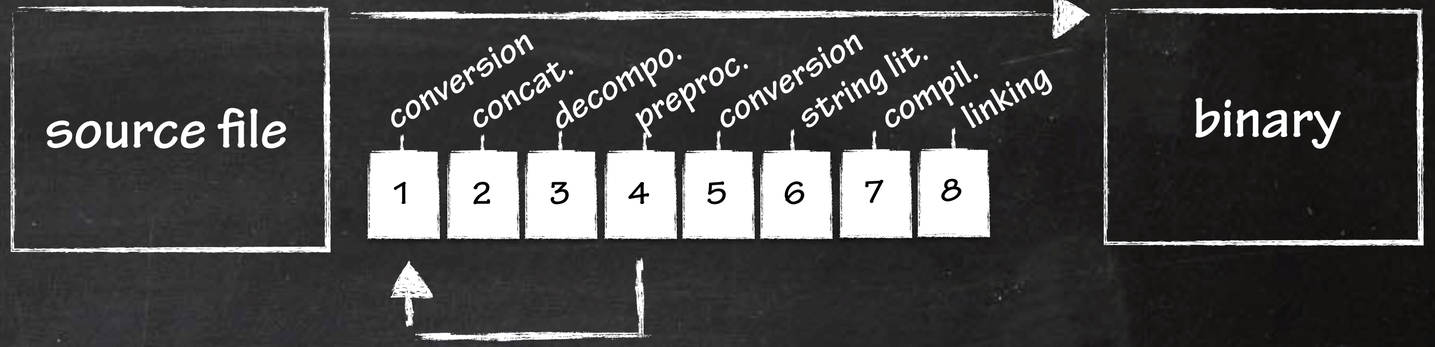
Phase 5

Conversion to Execution Character Set

- Conversion of characters & escape sequences in strings to execution character set
  - often multibyte character set such as UTF-8

based on: http://en.cppreference.com/w/c/language/translation_phases

Translation (4)



Phase 6

Phase 7

Phase 8

binary/executable

Concatenation of String Literals

"Hello" "World" becomes "Hello World"

Complication

- analyzing of tokens (syntactically and semantically)
- translation into translation unit

Linking

- translation units and libraries are bound to external references

based on: http://en.cppreference.com/w/c/language/translation_phases

Program Behavior

keep this in mind!

C language standard specifies observable behavior of programs…
… with the following exceptions in behavior:

- Undefined: it is not defined how the program behaves by illegal memory accesses out of bound, signed integer overflow, null pointer dereference,…
- Unspecified: more than one behaviors is possible at the same time, e.g., order of evaluation; result in valid results but may others when repeated
- Implementation-defined: implementation documents which behavior for unspecified behavior is chosen, e.g., bit number of byte
- Locale-specific: implementation-defined behavior that depends on locale, e.g., whether a given letter other than the 26 latin letters is uppercase or not

taken on: http://en.cppreference.com/w/c/language/behavior

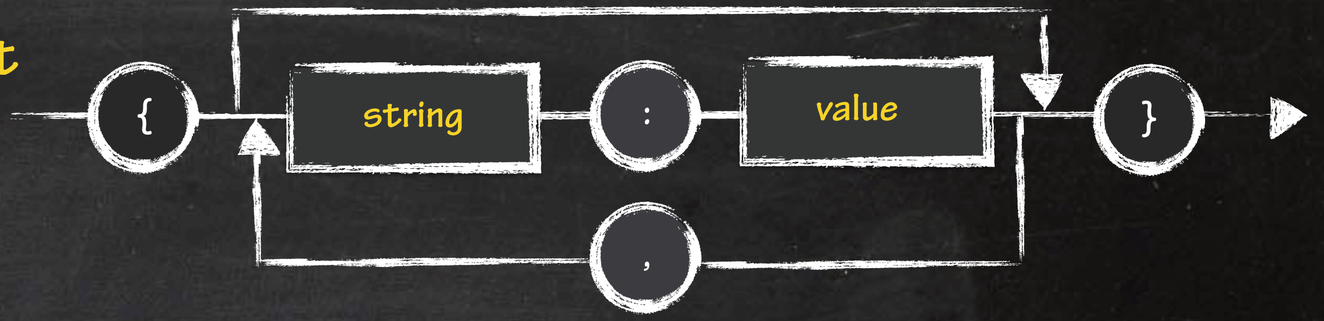
Intermezzo

Document-Based Storage and Querying

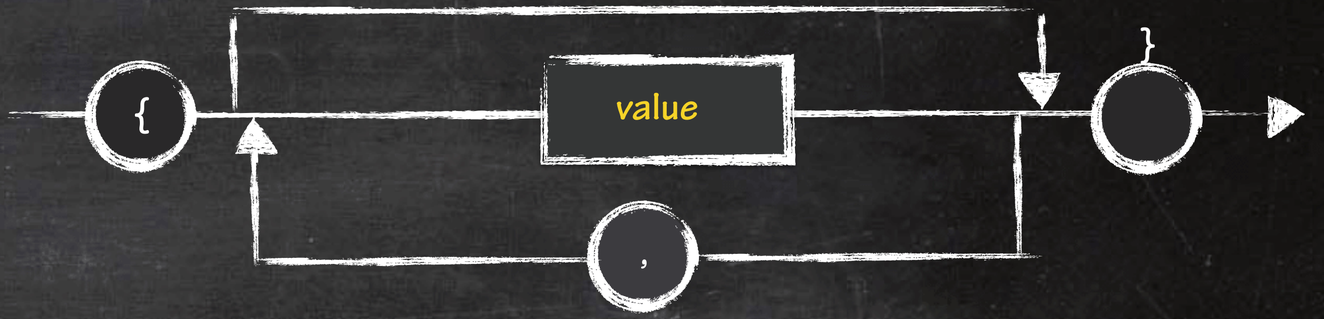
A Running Example

JavaScript Object Notation (JSON)

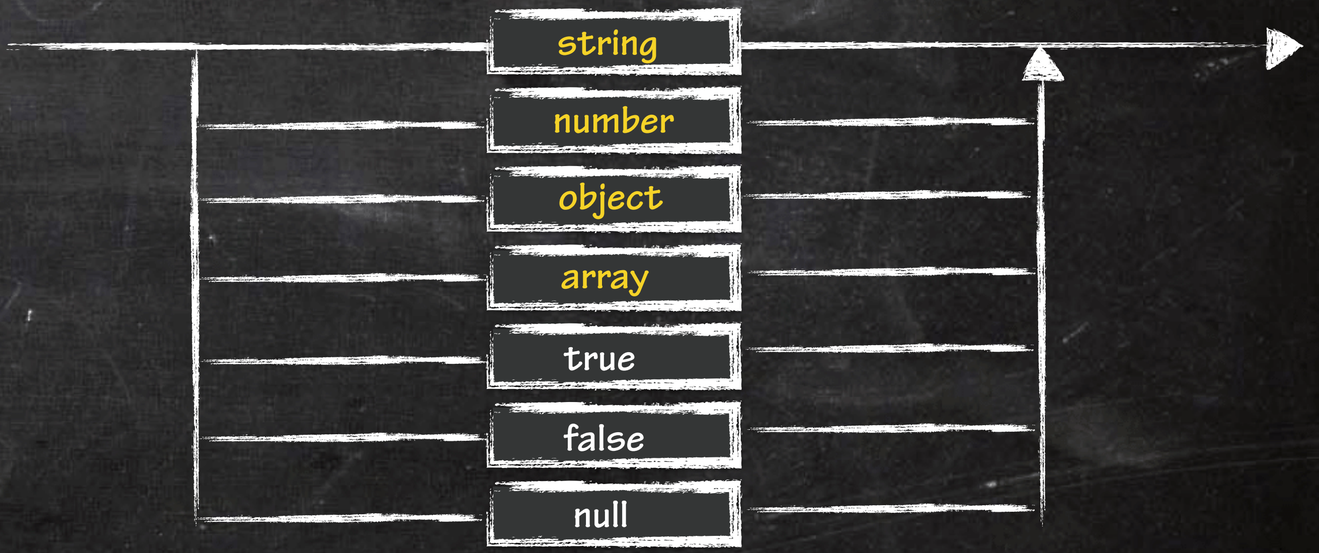
object



array



value



taken from: http://en.cppreference.com/w/c/header

Part II

Basic Concepts

- Comments
- Types
- Type Conversions
- Objects & Alignments
- Naming, Scoping & Lifetimes
- Lookup and Namespaces
- Storage Classes
- Keywords