CMPUT201W20B2 Week 12

Abram Hindle

March 31, 2020

Contents

1 Week12 **2**
- 1.1 Copyright Statement . . . 2
  - 1.1.1 License . . . 2
  - 1.1.2 Hazel Code is licensed under AGPL3.0+ . . . 2
- 1.2 Alternative version . . . 2
- 1.3 Init ORG-MODE . . . 2
  - 1.3.1 Org export . . . 3
  - 1.3.2 Org Template . . . 3
- 1.4 Remember how to compile? . . . 3
- 1.5 Numbers! . . . 3
  - 1.5.1 Binary . . . 4
  - 1.5.2 Octal . . . 5
  - 1.5.3 Base10 Review . . . 5
  - 1.5.4 Hex Review . . . 6
- 1.6 Floating Point Numbers . . . 7
  - 1.6.1 Resources . . . 7
  - 1.6.2 Single precision IEEE754 floating point numbers . . . 7
  - 1.6.3 Let's explore the bits of floats . . . 8
  - 1.6.4 How do I avoid typepunning? . . . 11
- 1.7 Modules . . . 12
- 1.8 Coupling . . . 12
  - 1.8.1 Coupling . . . 12
  - 1.8.2 Graph Coupling . . . 13
- 1.9 Cohesion . . . 25
  - 1.9.1 Refs . . . 26

1 Week12

https://github.com/abramhindle/CMPUT201W20B2-public/tree/master/week12

1.1 Copyright Statement

If you are in CMPUT201 at UAlberta this code is released in the public domain to you.

Otherwise it is (c) 2020 Abram Hindle, Hazel Campbell AGPL3.0+

1.1.1 License

CMPUT 201 C Notes Copyright (C) 2020 Abram Hindle, Hazel Campbell

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see https://www.gnu.org/licenses/.

1.1.2 Hazel Code is licensed under AGPL3.0+

Hazel's code is also found here https://github.com/hazelybell/examples/tree/C-2020-01

Hazel code is licensed: The example code is licensed under the AGPL3+ license, unless otherwise noted.

1.2 Alternative version

Checkout the .txt, the .pdf, and the .html version

1.3 Init ORG-MODE

```
;; I need this for org-mode to work well
;; If we have a new org-mode use ob-shell
;; otherwise use ob-sh --- but not both!
```

```
(if (require 'ob-shell nil 'noerror)
  (progn
    (org-babel-do-load-languages 'org-babel-load-languages '((shell . t))))
  (progn
    (require 'ob-sh)
    (org-babel-do-load-languages 'org-babel-load-languages '((sh . t)))))
(org-babel-do-load-languages 'org-babel-load-languages '((C . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((python . t)))
(setq org-src-fontify-natively t)
(setq org-confirm-babel-evaluate nil) ;; danger!
(custom-set-faces
 '(org-block ((t (:inherit shadow :foreground "black"))))
 '(org-code ((t (:inherit shadow :foreground "black")))))
```

1.3.1 Org export

```
(org-html-export-to-html)
(org-latex-export-to-pdf)
(org-ascii-export-to-ascii)
```

1.3.2 Org Template

Copy and paste this to demo C

```
#include <stdio.h>

int main(int argc, char**argv) {
    return 0;
}
```

1.4 Remember how to compile?

gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o programname programname.c

1.5 Numbers!

Computers love powers of 2 because we calculate everything via bits.

Bases used on computers:

2, 8, 10, 16

- 32 in base 16 is 2*pow(16,1) = 0x20

- 32 in base 10 is 3*pow(10,1) + 2*pow(10,0) = 3*10 + 2*1 = 32
- 32 in base 8 is 4*pow(8,1) = 4*8 = 32 = 040
- 32 in base 2 is 1*pow(2,5) = 0b100000
- 31 in base 16 is 1*pow(16,1) + 15*pow(16,0)
- 31 in base 10 is 3*pow(10,1) + 1*pow(10,0)
- 31 in base 8 is 3*pow(8,1) + 7*pow(8,0)
- 31 in base 2 is 1*pow(2,4) + 1*pow(2,3) + 1*pow(2,2) + 1*pow(2,1) + 1*pow(2,0)
- Notation for base 2 for 31: 0b11111 # not available in C, good in python
- Notation for base 8 for 31: 037 # available in C , good in python
- Notation for base 10 for 31: 31 # available in C , good in python
- Notation for base 16 for 31: 0x1F # available in C , good in python

1.5.1 Binary

```
Base 2: powers of 2

Digits: 0,1

0:  0b00000  8:  0b01000
1:  0b00001  9:  0b01001
2:  0b00010 10:  0b01010
3:  0b00011 11:  0b01011
4:  0b00100 12:  0b01100
5:  0b00101 13:  0b01101
6:  0b00110 14:  0b01110
7:  0b00111 15:  0b01111
            16:  0b10000
```

1.5.2 Octal

```
#include <stdio.h>

int main() {
   printf("%d\n", 037);
}
```

```
31
```

```
Base 8: powers of 8

3 bits

Digits: 0,1,2,3,4,5,6,7

0:  000  8: 010
1:  001  9: 011
2:  002 10: 012
3:  003 11: 013
4:  004 12: 014
5:  005 13: 015
6:  006 14: 016
7:  007 15: 017
        16: 020

07:                              7 =   7
077:                        7*8 + 7 =  63
0777:              7*8*8 + 7*8 + 7 = 511
07777: 7*8*8*8 + 7*8*8 + 7*8 + 7 = 4095
```

1.5.3 Base10 Review

```
Base 10: power of 10

   Digits: 0,1,2,3,4,5,6,7,8,9

~4 bits - not a power of 2

0:  0  8:  8
1:  1  9:  9
```

```
2:  2 10: 10
3:  3 11: 11
4:  4 12: 12
5:  5 13: 13
6:  6 14: 14
7:  7 15: 15
      16: 16
```

1.5.4 Hex Review

```
Base 16: power of 16

   Digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
or Digits: 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f

0:  0x00  8: 0x08
1:  0x01  9: 0x09
2:  0x02 10: 0x0A
3:  0x03 11: 0x0B
4:  0x04 12: 0x0C
5:  0x05 13: 0x0D
6:  0x06 14: 0x0E
7:  0x07 15: 0x0F
         16: 0x10
0xF    =                                    15 =    15 = 2^4 - 1
0xFF   =                             15*16 + 15 =   255 = 2^8 - 1
0xFFF  =                 15*16*16 + 15*16 + 15 =  4095 = 2^12 - 1
0xFFFF = 15*16*16*16 + 15*16*16 + 15*16 + 15 = 65535 = 2^16 - 1

Digit Lookup

0:  0x0 0b0000     8: 0x8 0b1000
1:  0x1 0b0001     9: 0x9 0b1001
2:  0x2 0b0010    10: 0xA 0b1010
3:  0x3 0b0011    11: 0xB 0b1011
4:  0x4 0b0100    12: 0xC 0b1100
5:  0x5 0b0101    13: 0xD 0b1101
6:  0x6 0b0110    14: 0xE 0b1110
7:  0x7 0b0111    15: 0xF 0b1111
```

```
1 nibble = 4 bits = 1 hex digit
```

1.6 Floating Point Numbers

1.6.1 Resources

Helpful resources:

- `https://en.wikipedia.org/wiki/IEEE_754`
- `http://steve.hollasch.net/cgindex/coding/ieeefloat.html`

1.6.2 Single precision IEEE754 floating point numbers

Type: float
-Sign: 0 - positive, 1 negative -Exponent: unsigned 8 bit integer with 127 (2^7-1) offset

- -126 to 127
- Fraction: 24 bits (only 23 used). The first bit is implied.
  - So 00000000000000000000000 is actually 10000000000000000000000

Sign (1 bit)	Exponent (8 bit)	b10	Fraction (23 bit)	Value	base 2
0	01111100	-3	01000000000000000000000	0.15625	1/8 + 1/32
1	01111100	-3	01000000000000000000000	-0.15625	
0	01111100	-3	01000000000000000000000	-0.15625	
0	00000000	0	00000000000000000000000	0	
1	00000000	0	00000000000000000000000	-0	
0	11000000	65	00000000000000000000000	3.68e+19	2^{65}
1	11111111	0	00000000000000000000000	-inf	
0	11111111	0	00000000000000000000000	inf	
0	11111111	0	10000000000000000000000	nan	
0	11111111	0	00000000000000000000001	nan	

- Neat design trick: you can sort them as signed integers (two's complement)!
- -0 is right before 0 for signed integer sorting

0 01111100 -3 01000000000000000000000 0.15625 1/8 + 1/32

0.15625 to floating point. Implied bit is pow(2,-3)
pow(2,-3) + 0*pow(2,-4) + 1*pow(2,-5) (pow(2,5)==32, pow(2,-5) = 1/32.0)

1.6.3 Let's explore the bits of floats

```
#include <math.h>
#include <stdio.h>
#include <inttypes.h>
// 0x1L MUST be used 0x1 causes bugs
#define BIT(x,y) (x & (0x1L << y))
#define MAXBITSTRBITS 129
static char _bitstr[MAXBITSTRBITS] = { '\0' };
static char * bitString(uint64_t value, const unsigned int bits) {
    // iterator must be uint64_t
    for (uint64_t i = 0 ; i < bits; i++) {
        char bit = (BIT(value,i))?'1':'0';
        _bitstr[bits-1-i] = bit;
    }
    _bitstr[bits] = '\0';
    return _bitstr;
}

// NOT PORTABLE
struct float_t {
    unsigned int mantissa:23; // LOWEST
    unsigned int exponent:8;
    unsigned int sign:1; // HIGHEST
};
// type pun for fun!
union floatint {
    float f;
    uint32_t i;
    struct float_t t;
};


int main() {
    float floats[] = {
        0.0,
       -0.0,
        INFINITY,
       -INFINITY,
        NAN,
```

```
    0.00001,
    0.0001,
    0.001,
    0.01,
    0.1,
    1.0,
    1/64.0,
    1/32.0,
    1/16.0,
    1/8.0,
    1/4.0,
    1/2.0,
    1.0,
    2.0,
    4.0,
    128.0,
    65536.0,
    0.15625,
    -0.15625,
    36893488147419103232.0, // 2**65
    -36893488147419103232.0 // 2**65
};
size_t nfloats = sizeof(floats)/sizeof(float);
printf("We're printing floats!\n");
printf("sizeof(floatint) == %lu\n", sizeof(union floatint));
printf("sizeof(float_t) == %lu\n", sizeof(struct float_t));
for (size_t i = 0; i < nfloats; i++) {
    float f = floats[i];
    union floatint fi = {.f=f};
    printf("%12g 0x%08x 0b%s\n", f, fi.i, bitString(fi.i, 32));
    printf("%12g 0x%08x sign: %hhu exponent: %3hhu exp-127: %4d mantissa: 0x%06x\n"
           f,
           fi.i,
           fi.t.sign,
           fi.t.exponent,
           (int)fi.t.exponent - (int)127,
           fi.t.mantissa
    );
    /* // We don't need bitfields
    printf("%12g %08x sign: %hhu exponent: %hhu exp-127: %4d mantissa: 0x%06x\n",
```

```
                f,
                fi.i,
                (fi.i >> 31),
                ((fi.i << 1) >> 24),
                (int)((fi.i << 1) >> 24) - (int)127,
                (fi.i & 0x007FFFFF)
            );
            */

        }
        puts("");
}

We're printing floats!
sizeof(floatint) == 4
sizeof(float_t) == 4
           0 0x00000000 0b00000000000000000000000000000000
           0 0x00000000 sign: 0 exponent:   0 exp-127: -127 mantissa: 0x000000
          -0 0x80000000 0b10000000000000000000000000000000
          -0 0x80000000 sign: 1 exponent:   0 exp-127: -127 mantissa: 0x000000
         inf 0x7f800000 0b01111111100000000000000000000000
         inf 0x7f800000 sign: 0 exponent: 255 exp-127:  128 mantissa: 0x000000
        -inf 0xff800000 0b11111111100000000000000000000000
        -inf 0xff800000 sign: 1 exponent: 255 exp-127:  128 mantissa: 0x000000
         nan 0x7fc00000 0b01111111110000000000000000000000
         nan 0x7fc00000 sign: 0 exponent: 255 exp-127:  128 mantissa: 0x400000
       1e-05 0x3727c5ac 0b00110111001001111100010110101100
       1e-05 0x3727c5ac sign: 0 exponent: 110 exp-127:  -17 mantissa: 0x27c5ac
      0.0001 0x38d1b717 0b00111000110100011011011100010111
      0.0001 0x38d1b717 sign: 0 exponent: 113 exp-127:  -14 mantissa: 0x51b717
       0.001 0x3a83126f 0b00111010100000110001001001101111
       0.001 0x3a83126f sign: 0 exponent: 117 exp-127:  -10 mantissa: 0x03126f
        0.01 0x3c23d70a 0b00111100001000111101011100001010
        0.01 0x3c23d70a sign: 0 exponent: 120 exp-127:   -7 mantissa: 0x23d70a
         0.1 0x3dcccccd 0b00111101110011001100110011001101
         0.1 0x3dcccccd sign: 0 exponent: 123 exp-127:   -4 mantissa: 0x4ccccd
           1 0x3f800000 0b00111111100000000000000000000000
           1 0x3f800000 sign: 0 exponent: 127 exp-127:    0 mantissa: 0x000000
   0.015625 0x3c800000 0b00111100100000000000000000000000
   0.015625 0x3c800000 sign: 0 exponent: 121 exp-127:   -6 mantissa: 0x000000
```

```
    0.03125 0x3d000000 0b00111101000000000000000000000000
    0.03125 0x3d000000 sign: 0 exponent: 122 exp-127:   -5 mantissa: 0x000000
     0.0625 0x3d800000 0b00111101100000000000000000000000
     0.0625 0x3d800000 sign: 0 exponent: 123 exp-127:   -4 mantissa: 0x000000
      0.125 0x3e000000 0b00111110000000000000000000000000
      0.125 0x3e000000 sign: 0 exponent: 124 exp-127:   -3 mantissa: 0x000000
       0.25 0x3e800000 0b00111110100000000000000000000000
       0.25 0x3e800000 sign: 0 exponent: 125 exp-127:   -2 mantissa: 0x000000
        0.5 0x3f000000 0b00111111000000000000000000000000
        0.5 0x3f000000 sign: 0 exponent: 126 exp-127:   -1 mantissa: 0x000000
          1 0x3f800000 0b00111111100000000000000000000000
          1 0x3f800000 sign: 0 exponent: 127 exp-127:    0 mantissa: 0x000000
          2 0x40000000 0b01000000000000000000000000000000
          2 0x40000000 sign: 0 exponent: 128 exp-127:    1 mantissa: 0x000000
          4 0x40800000 0b01000000100000000000000000000000
          4 0x40800000 sign: 0 exponent: 129 exp-127:    2 mantissa: 0x000000
        128 0x43000000 0b01000011000000000000000000000000
        128 0x43000000 sign: 0 exponent: 134 exp-127:    7 mantissa: 0x000000
      65536 0x47800000 0b01000111100000000000000000000000
      65536 0x47800000 sign: 0 exponent: 143 exp-127:   16 mantissa: 0x000000
    0.15625 0x3e200000 0b00111110001000000000000000000000
    0.15625 0x3e200000 sign: 0 exponent: 124 exp-127:   -3 mantissa: 0x200000
   -0.15625 0xbe200000 0b10111110001000000000000000000000
   -0.15625 0xbe200000 sign: 1 exponent: 124 exp-127:   -3 mantissa: 0x200000
3.68935e+19 0x60000000 0b01100000000000000000000000000000
3.68935e+19 0x60000000 sign: 0 exponent: 192 exp-127:   65 mantissa: 0x000000
-3.68935e+19 0xe0000000 0b11100000000000000000000000000000
-3.68935e+19 0xe0000000 sign: 1 exponent: 192 exp-127:   65 mantissa: 0x000000
```

1.6.4 How do I avoid typepunning?

The C99 is OK with union type punning :-/

The "blessed" way that is C11 compatible is memcpy.

Bitfields are generally considered "CURSED" due to platform specific behaviour.

You should only memcpy direct types like uint64_t and double IFF they have the same size.

```
// PUN64 copies the bits of double value into a 64bit unsigned int and
// returns it.
```

```
uint64_t PUN64(double value) {
  uint64_t output = 0;
  memcpy(&output, &value, sizeof(uint64_t));
  return output;
}
```

1.7 Modules

A module can have different levels of granularity:

- function level
- file level (.c or .h)
- library level (shared code .so)
- process level (an executable)
- platform level (GNU/Linux)
- service level (distributed systems, webservices)
- ecosystem level (Ubuntu or pip or web)

In C typically a module refers to a pair of .c file or a pair of .h and .c files.

1.8 Coupling

Coupling often refers to how much 1 module (like a .c file or a .h file) relies on other modules.

1.8.1 Coupling

1 fast way in C of calculating coupling is to count the number of include directives.

The number of includes gives one an idea of how many modules that your module relies upon.

In object oriented languages like Java or C++ coupling often refers to the number of modules (Classes) that a single class refers to.

Coupling is related to dependencies. Too many dependencies makes a system complicated–often erroneously called complex. If one of these dependencies changes, breaks, is deprecated it can harm anything that relies

upon it. Knowing that a module is heavily relied upon is important as well because it implies the risk involved with changing that module.

High coupling is bad because it makes code weak to change:

- too many dependencies increase change of change
- too many dependencies increase bug risk
- too many dependencies means dependencies will probably change

In "structured design" as suggested by Constantine there are multiple kinds of coupling.

- Data coupling: you pass parameters to another module
  - connected by name
- Control coupling: you use another module for control flow
- Content/Common coupling: when you use code directly from another module
  - macros can cause this problem
  - includes could do this if it includes code.

Fundamentally in C at the file level we're really just going to be using data coupling. If you say coupling most people will assume this is what you are talking about. Essentially your .h or .c file refers to other modules and their parts by name.

1.8.2 Graph Coupling

From Pressman and Dhama [DHA95].

Model your modules as a directed graph.

- Modules are vertices.
- References are edges.

Fan-out is the number of modules a module calls.

Fan-in is the number of modules that call that module. Fan-in does not imply the module is truly coupled to other modules, just that it is popular and relied upon. Meaning it shouldn't change much!

From Pressman and Dhama [DHA95]:

For data and control flow coupling,

- d_i = number of input data parameters
- c_i = number of input control parameters
- d_o = number of output data parameters
- c_o = number of output control parameters

For global coupling,

- g_d = number of global variables used as data
- g_c = number of global variables used as control

For environmental coupling,

- w = number of modules called (fan-out)
- r = number of modules calling the module under consideration (fan-in)

Using these measures, a module coupling indicator, m c , is defined in the following way:

- $m_c = k/M$
- where k = 1, a proportionality constant 8 and
- $M = d_i + (a * c_i) + d_o + (b * c_o) + g_d + (c * g_c) + w + r$
- where a = b = c = 2.

High m_c: low coupling Low m_c: high coupling

1. Coupling_h – my take

   My take?

   Forget about the nitty gritty coupling and focus on fan-out and dependencies.

   I would calculate coupling as unidirectional:

   - where x = number of unique external function calls
   - where y = number of unique external variable access
   - where z = number of unique externally defined types

- where lambda ~ 4 or however more important you feel that fan-out is
- w = number of modules called (fan-out)
- coupling_h = 1 / (1 + lambda*w + x + y + z)

If coupling_h is 1 you are decoupled.

If coupling_h in near 0.0 you are pretty heavily coupled.

For a C module I recommend counting both .h and .c together

You can invent your own coupling measures for your own purposes.

2. Low coupling

   (a) No Coupling

   ```
   // 0 coupling, no modules
   #define RET 404
   int main() {
       return RET;
   }
   ```

   - w = x = y = z = 0
   - lamba = 4
   - coupling_h = 1/(1+0) = 1

   (b) A little coupling
   We're now related to the stdlib IO routines

   ```
   #include <stdio.h>
   #define RET
   int main() {
       printf("%d\n", RET);
       return RET;
   }
   ```

   - w = 1
   - x = 1
   - y = z = 0
   - lamba = 4
   - coupling_h = 1/(4*1+1+0+0) = 1/5

   (c) A little more coupling
   Now we're bound to stdlib.h and randr

```
#include <stdio.h>
#include <stdlib.h>
#define RET
int main() {
    printf("%d\n", rand());
    return RET;
}
```

- w = 2
- x = 2
- y = z = 0
- lamba = 4
- $\text{coupling}_{\text{h}} = 1/(4*2+2+0+0) = 1/10$

(d) More coupling

Now we're bound to string.h as well

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define RET
int main() {
    char * str1 = "aaaaaa";
    char * str2 = "aaaaab";
    printf("%d\n", rand());
    printf("%d\n", strcmp(str1,str2));
    return RET;
}
```

- w = 3
- x = 3
- y = z = 0
- lamba = 4
- $\text{coupling}_{\text{h}} = 1/(4*3+3+0+0) = 1/15$

(e) Even More Coupling:

```
/*

                                    Encode CSV file into MIDI
```

```
        The CSV file must be in the format generated by midicsv--while
        you can add and delete events and change their contents, the
        overall organisation of the file must be the same.

        Designed and implemented in October of 1998 by John Walker.
        Revised and updated in February of 2004 by the same perp.

                        http://www.fourmilab.ch/

                This program is in the public domain.

*/

#define PROG    "csvmidi"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#ifdef _WIN32
#include <fcntl.h>
#include <io.h>
#define strcasecmp _stricmp
#endif

#include "version.h"
#include "types.h"
#include "midifile.h"
#include "midio.h"
#include "csv.h"
#include "getopt.h"
```

- w = 10 or 12
- x = ??
- y = z = ??
- lamba = 4
- $\text{coupling}_{\text{h}} = 1/(4*10+0+0) = 1/40$

(f) A lot of coupling
GUI apps often rely on lots of libraries.

Here's the gnome evince PDF reader.
It needs:

- GUI
- PDF
- Compression
- ???

```
ldd /usr/bin/evince
```

```
linux-vdso.so.1 (0x00007ffd69982000)
libevdocument3.so.4 => /usr/lib/x86_64-linux-gnu/libevdocument3.so.4 (0x00007
libevview3.so.3 => /usr/lib/x86_64-linux-gnu/libevview3.so.3 (0x00007f3c99ec7
libsecret-1.so.0 => /usr/lib/x86_64-linux-gnu/libsecret-1.so.0 (0x00007f3c99c
libgnome-desktop-3.so.17 => /usr/lib/x86_64-linux-gnu/libgnome-desktop-3.so.1
libgtk-3.so.0 => /usr/lib/x86_64-linux-gnu/libgtk-3.so.0 (0x00007f3c99133000)
libgdk-3.so.0 => /usr/lib/x86_64-linux-gnu/libgdk-3.so.0 (0x00007f3c98e3d000)
libpangocairo-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpangocairo-1.0.so.0 (0
libpango-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpango-1.0.so.0 (0x00007f3c98
libatk-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libatk-1.0.so.0 (0x00007f3c987bd0
libcairo-gobject.so.2 => /usr/lib/x86_64-linux-gnu/libcairo-gobject.so.2 (0x00
libcairo.so.2 => /usr/lib/x86_64-linux-gnu/libcairo.so.2 (0x00007f3c98297000)
libgio-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgio-2.0.so.0 (0x00007f3c97ef80
libgdk_pixbuf-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgdk_pixbuf-2.0.so.0 (0
libgobject-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgobject-2.0.so.0 (0x00007
libglib-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libglib-2.0.so.0 (0x00007f3c9770
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f3c973cb000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f3c971ac000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f3c96dbb000)
libgmodule-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libgmodule-2.0.so.0 (0x00007
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007f3c9699a000)
libgstvideo-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libgstvideo-1.0.so.0 (0x000
libgstreamer-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libgstreamer-1.0.so.0 (0x00
libgcrypt.so.20 => /lib/x86_64-linux-gnu/libgcrypt.so.20 (0x00007f3c960aa000)
libX11.so.6 => /usr/lib/x86_64-linux-gnu/libX11.so.6 (0x00007f3c95d72000)
libudev.so.1 => /lib/x86_64-linux-gnu/libudev.so.1 (0x00007f3c95b54000)
libseccomp.so.2 => /lib/x86_64-linux-gnu/libseccomp.so.2 (0x00007f3c9590d000)
librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f3c95705000)
libXi.so.6 => /usr/lib/x86_64-linux-gnu/libXi.so.6 (0x00007f3c954f5000)
libXfixes.so.3 => /usr/lib/x86_64-linux-gnu/libXfixes.so.3 (0x00007f3c952ef000
libatk-bridge-2.0.so.0 => /usr/lib/x86_64-linux-gnu/libatk-bridge-2.0.so.0 (0
```

```
libepoxy.so.0 => /usr/lib/x86_64-linux-gnu/libepoxy.so.0 (0x00007f3c94dbd000)
libpangoft2-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libpangoft2-1.0.so.0 (0x000
libfontconfig.so.1 => /usr/lib/x86_64-linux-gnu/libfontconfig.so.1 (0x00007f3
libXinerama.so.1 => /usr/lib/x86_64-linux-gnu/libXinerama.so.1 (0x00007f3c947
libXrandr.so.2 => /usr/lib/x86_64-linux-gnu/libXrandr.so.2 (0x00007f3c9455400
libXcursor.so.1 => /usr/lib/x86_64-linux-gnu/libXcursor.so.1 (0x00007f3c9434a
libXcomposite.so.1 => /usr/lib/x86_64-linux-gnu/libXcomposite.so.1 (0x00007f3
libXdamage.so.1 => /usr/lib/x86_64-linux-gnu/libXdamage.so.1 (0x00007f3c93f44
libxkbcommon.so.0 => /usr/lib/x86_64-linux-gnu/libxkbcommon.so.0 (0x00007f3c9
libwayland-cursor.so.0 => /usr/lib/x86_64-linux-gnu/libwayland-cursor.so.0 (0
libwayland-egl.so.1 => /usr/lib/x86_64-linux-gnu/libwayland-egl.so.1 (0x00007
libwayland-client.so.0 => /usr/lib/x86_64-linux-gnu/libwayland-client.so.0 (0
libXext.so.6 => /usr/lib/x86_64-linux-gnu/libXext.so.6 (0x00007f3c934da000)
libfreetype.so.6 => /usr/lib/x86_64-linux-gnu/libfreetype.so.6 (0x00007f3c932
libthai.so.0 => /usr/lib/x86_64-linux-gnu/libthai.so.0 (0x00007f3c9301d000)
libpixman-1.so.0 => /usr/lib/x86_64-linux-gnu/libpixman-1.so.0 (0x00007f3c92d
libpng16.so.16 => /usr/lib/x86_64-linux-gnu/libpng16.so.16 (0x00007f3c92b46000
libxcb-shm.so.0 => /usr/lib/x86_64-linux-gnu/libxcb-shm.so.0 (0x00007f3c92943
libxcb.so.1 => /usr/lib/x86_64-linux-gnu/libxcb.so.1 (0x00007f3c9271b000)
libxcb-render.so.0 => /usr/lib/x86_64-linux-gnu/libxcb-render.so.0 (0x00007f3
libXrender.so.1 => /usr/lib/x86_64-linux-gnu/libXrender.so.1 (0x00007f3c92304
libselinux.so.1 => /lib/x86_64-linux-gnu/libselinux.so.1 (0x00007f3c920dc000)
libresolv.so.2 => /lib/x86_64-linux-gnu/libresolv.so.2 (0x00007f3c91ec1000)
libmount.so.1 => /lib/x86_64-linux-gnu/libmount.so.1 (0x00007f3c91c6d000)
libffi.so.6 => /usr/lib/x86_64-linux-gnu/libffi.so.6 (0x00007f3c91a65000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007f3c917f3000)
/lib64/ld-linux-x86-64.so.2 (0x00007f3c9a5c4000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f3c915ef000)
libgstbase-1.0.so.0 => /usr/lib/x86_64-linux-gnu/libgstbase-1.0.so.0 (0x00007
liborc-0.4.so.0 => /usr/lib/x86_64-linux-gnu/liborc-0.4.so.0 (0x00007f3c910fe
libgpg-error.so.0 => /lib/x86_64-linux-gnu/libgpg-error.so.0 (0x00007f3c90ee9
libdbus-1.so.3 => /lib/x86_64-linux-gnu/libdbus-1.so.3 (0x00007f3c90c9c000)
libatspi.so.0 => /usr/lib/x86_64-linux-gnu/libatspi.so.0 (0x00007f3c90a6c000)
libharfbuzz.so.0 => /usr/lib/x86_64-linux-gnu/libharfbuzz.so.0 (0x00007f3c907
libexpat.so.1 => /lib/x86_64-linux-gnu/libexpat.so.1 (0x00007f3c9059c000)
libdatrie.so.1 => /usr/lib/x86_64-linux-gnu/libdatrie.so.1 (0x00007f3c9039500
libXau.so.6 => /usr/lib/x86_64-linux-gnu/libXau.so.6 (0x00007f3c90191000)
libXdmcp.so.6 => /usr/lib/x86_64-linux-gnu/libXdmcp.so.6 (0x00007f3c8ff8b000)
libblkid.so.1 => /lib/x86_64-linux-gnu/libblkid.so.1 (0x00007f3c8fd3e000)
libsystemd.so.0 => /lib/x86_64-linux-gnu/libsystemd.so.0 (0x00007f3c8faba000)
```

```
libgraphite2.so.3 => /usr/lib/x86_64-linux-gnu/libgraphite2.so.3 (0x00007f3c8
libbsd.so.0 => /lib/x86_64-linux-gnu/libbsd.so.0 (0x00007f3c8f678000)
libuuid.so.1 => /lib/x86_64-linux-gnu/libuuid.so.1 (0x00007f3c8f471000)
liblzma.so.5 => /lib/x86_64-linux-gnu/liblzma.so.5 (0x00007f3c8f24b000)
liblz4.so.1 => /usr/lib/x86_64-linux-gnu/liblz4.so.1 (0x00007f3c8f02f000)
```

- w ~ 70
- x = ??
- y = z = ??
- lamba = 4
- $\text{coupling}_{\text{h}} = 1/(4*70+0+0) = 1/280$

(g) More examples from Assignment 8

```
// Have a guard to ensure that we don't include it multiple times.
#ifndef _CHECKINPUT_H_
/* checkInput: given the result of scanf check if it
 * 0 elements read or EOF. If so exit(1) with a warning.
 *
 */
#define _CHECKINPUT_H_
void checkInput(int err); // a prototype!
#endif
```

- $\text{coupling}_{\text{h}} = ?$

```
#include "parameters.h"
#include <stdio.h>
#include <stdlib.h>

// You must implement a guard

struct combo_t;

typedef struct combo_t * Combo;

// Create a Cartesian Product Combo generator
Combo createParameterCombo();
// Add parameter definition to the Combo
void addParameterDefCombo(Combo combo, ParameterDef def);
// Add a parameter of a previous parameter definition to the Combo
size_t addParameterCombo(Combo combo, Parameter param);
```

```
// How man parameter definitions are defined in this combo
size_t nParamsCombo(Combo combo);
// a boolean of whether or not there are more product combos to come
// used for terminating while loops
bool hasNextCombo(Combo combo);
// The next product combo as a malloc'd array of parameters of
// nParamsCombo(combo) length
Parameter * nextCombo(Combo combo);
// Free a parameter list created by nextCombo
void freeParamsCombo(Combo combo, Parameter * params);
// Free a cartesian product combo generator
void freeCombo(Combo combo);
```

- w=3
- z=2
- $\text{coupling}_{\text{h}} = 1/(4*\text{w} + \text{z})$

```
// Implement a guard

#include <inttypes.h>
#include <stdbool.h>
#include <stdlib.h>

enum type_flag {
  LONG,
  DOUBLE,
  STRING,
  CHAR,
};

typedef enum type_flag TypeFlag;

union any_t {
  int64_t aLong;
  double aDouble;
  char * aString;
  char aChar;
};

typedef union any_t Any;
```

```
struct parameter_def {
  char * name;
  TypeFlag type;
};

typedef struct parameter_def ParameterDef;

struct parameter_t {
  ParameterDef def;
  Any any;
};

typedef struct parameter_t Parameter;

// create a parameter def for aDouble
ParameterDef doubleParameterDef( char * name );
// create a parameter def for aLong
ParameterDef longParameterDef(   char * name );
// create a parameter def for aString
ParameterDef stringParameterDef( char * name );
// create a parameter def for aChar
ParameterDef charParameterDef(   char * name );
// create a parameter def for aChar
ParameterDef mkParameterDef(   char * name, TypeFlag flag );

// Are param1 and param2 equal?
bool         equalParameterDef(  ParameterDef param1, ParameterDef param2);

// create a double parameter
Parameter    mkDoubleParameter(  ParameterDef def, double value);
// create a long parameter
Parameter    mkLongParameter(    ParameterDef def, long   value);
// create a string parameter
Parameter    mkStringParameter(  ParameterDef def, char * value);
// create a char parameter
Parameter    mkCharParameter(    ParameterDef def, char   value);
// return the name of a parameter's def
char *       nameParameter(      Parameter param );
// return the type of a parameter from its def
```

```
TypeFlag     typeParameter(      Parameter param );
// return the double value of a parameter
double       doubleParameter(    Parameter param );
// return the long value of a parameter
long         longParameter(      Parameter param );
// return the string value of a parameter
char *       stringParameter(    Parameter param );
// return the char value of a parameter
char         charParameter(      Parameter param );
// Are param1 and param2 equal?
bool         equalParameter(     Parameter param1, Parameter param2);
// Print parameters!
void printParams(Parameter * params, size_t size);
```

3 includes & 1 reference to int64_{t}

- w=3
- z=1
- $\text{coupling}_{\text{h}} = 1/(4*\text{w} + \text{z})$

```
#include "parameters.h"
#include <inttypes.h>
#include <stdlib.h>
#include <stdio.h>

// You need a guard here!

// forward declaration
union ppd_t;
// This is an array of defs, params, or arrays
// Assume you keep the same sort of values within this array
struct ppd_array {
  union ppd_t * params; // A pointer to an array of ppd_t unions (PPD)
  size_t size;
};
typedef struct ppd_array * PPDArray;

union ppd_t {
  ParameterDef       def;
  Parameter          param;
  struct ppd_array * array;
```

```
  int64_t            int64;
  char             * str;
};

typedef union ppd_t PPD;

// create PPDArray
PPDArray createPPDArray();
// get the number of elements in a PPDArray
size_t sizePPDArray(PPDArray array);
// Free a PPD Array
void freePPDArray(PPDArray array);
// Free a PPD Array and all of its PPD Arrays (not recursive)
// Frees an array of arrays, by freeing each contained array
void freeArrayPPDArray(PPDArray array);
// return a parameter def at index of PPDArray array
ParameterDef parameterDefPPDArray(PPDArray array, size_t index);
// return a pointer to parameter def at index of PPDArray array
// pretty unsafe
ParameterDef * parameterDefRefPPDArray(PPDArray array, size_t index);
// return a parameter at index of PPDArray array
Parameter parameterPPDArray(PPDArray array, size_t index);
// return a array at index of PPDArray array
PPDArray arrayPPDArray(PPDArray array, size_t index);
// return an int64 at index of PPDArray array
int64_t int64PPDArray(PPDArray array, size_t index);
// return a string at index of PPDArray array
char * stringPPDArray(PPDArray array, size_t index);
// at index of array set the int64 value to
void setInt64PPDArray(PPDArray array, size_t index, int64_t value);
// extend PPDArray 1 slot and put this PPD into that slot
// may use realloc!
void addPPDPPDArray(PPDArray array, PPD defOrParam);
// extend PPDArray 1 slot and put this ParameterDef into that slot
// may use realloc!
void addDefPPDArray(PPDArray array, ParameterDef def);
// extend PPDArray 1 slot and put this Parameter into that slot
// may use realloc!
void addParamPPDArray(PPDArray array, Parameter param);
// extend PPDArray 1 slot and put this Array newArray into that slot
```

```
// may use realloc!
void addPPDArrayPPDArray(PPDArray array, PPDArray newArray  );
// extend PPDArray 1 slot and put this int64 into that slot
// may use realloc!
void addInt64PPDArray(PPDArray array, int64_t int64  );
// extend PPDArray 1 slot and put this string into that slot
// may use realloc!
void addStringPPDArray(PPDArray array, char * str );
```

3 includes & 3 reference to int64_t, Parameter and ParameterDef

- w=3
- z=3
- $\text{coupling}_\text{h} = 1/(4*\text{w} + \text{z})$

1.9 Cohesion

Cohesion is about how relevant code is to itself. A highly cohesive module is topical, that is about it is about a single topic.

stdlib.h is low cohesion because it does too many things. From memory allocation to random number generation stdlib.h does too much. It is not cohesive.

math.h has better cohesion than stdlib.h but its cohesion is considered low because the functions are just logically related to math and floating point operations. In fact many of the operations could be refactored and moved into floating point specific libraries.

parameters.h is very cohesive, it only deals with parameters and parameter defs. Its responsibilities are the creation of parameters and parameter defs. Its only weird responsibility is for printing parameters which is probably its least cohesive part.

High cohesion is good because it means the module is topical and relevant. It doesn't have any weird responsibilities. It won't have to change to address extra responsibilities. High cohesion typically limits coupling.

To calculate cohesion there's many metrics suggested but I suggest reading the module and writing bullet points about responsibilities. Let's look at ppdarray.c:

- representation: structs for PPD array and containers
- creation & allocation
- destruction & deallocation

- array access
- array appending

So 5 responsibilities. Let's look at checkinput.h:

- error detection
- error reporting

Checkinput.h is clearly more cohesive than PPDArray mostly because it has less responsibilities. Even though both are very cohesive modules

My stdio.h has 122 externs defined within it. It is way less cohesive than PPDArray but all of stdio.h's externs are about io to terminals and files. So it is still logically cohesive.

My stdlib.h has 153 externs defined within it. But it handles memory allocation, temporary files, environment variables, sorting, binary search, converting floats to strings, etc. No cohesive and often called coincidental cohesion (aka you're lucky if it is cohesive).

1.9.1 Refs

W. P. Stevens, G. J. Myers and L. L. Constantine, "Structured design," in IBM Systems Journal, vol. 13, no. 2, pp. 115-139, 1974. `https://ieeexplore.ieee.org/document/5388187`

Roger Pressman. 2009. Software Engineering: A Practitioner's Approach (7th. ed.). McGraw-Hill, Inc., USA.

[DHA95] Dhama, H., "Quantitative Models of Cohesion and Coupling in Software," Journal of Systems and Software, vol. 29, no. 4, April 1995.