```
/*
##########################################################################
##########################################################################
#<a rel="license" href="http://creativecommons.org/licenses/by-nc-
nd/4.0/"><img alt="Creative Commons Licence" style="border-width:0"
src="https://i.creativecommons.org/l/by-nc-nd/4.0/88x31.png" /></a><br
/>This work is licensed under a <a rel="license"
href="http://creativecommons.org/licenses/by-nc-nd/4.0/">Creative Commons
Attribution-NonCommercial-NoDerivatives 4.0 International License</a>.
##########################################################################
##########################################################################
#Course Authored By:
#-------------------------------------------------------------------------
----------------------------------
#K.Srinivasan
#NeuronRain Documentation and Licensing: http://neuronrain-
documentation.readthedocs.io/en/latest/
#Personal website(research): https://sites.google.com/site/kuja27/
#-------------------------------------------------------------------------
----------------------------------
##########################################################################
##########################################################################
*/

This is a non-linearly organized, code puzzles oriented, continually
updated set of course notes on C language. Though C is at present
rapidly diminishing in usage because of many high level languages and big
data tools for cloud processing, a primer in C is essential for
grasping nuances of operating system memory management, networking and
threading
-------------------------------------------------------------------------
---------------------------------------
References:
-----------
1. C Puzzle Book - Alan R.Feuer
2. C Programming Language - ANSI C - [Kernighan-Ritchie]

2 February 2017
---------------
Arithmetic Operators:

#include <stdio.h>

main()
{
        int x;
        x = -3 + 4 * 5 - 6; printf("%d\n",x);
        x = 3 + 4 % 5 - 6; printf("%d\n",x);
        x = -3 * 4 % -6 / 5; printf("%d\n",x);
        x = (7 + 6) % 5 / 2; printf("%d\n",x);
}

/*
        Above prints:
        11 (-3 unary operator has precedence over binary, * has
precedence over + and -. (-3)+(20) -6 = 11 )
        1 ( remainder operator (modulus) % has precedence. 3 + (4) - 6 =
1)
        0 ( (((-3 * 4) % -6)/5) = (-12 % -6) % (-1) = 0. Operator
precedence is same and evaluation is from left to right )
```

```
        1 ( ((13) % 5) / 2 = 3 % 2 = 1 )
*/
```

---

10 March 2017

---

Pointers in C are basic low level primitives of allocating memory and indexing them by contiguous bytes (malloc/free). Most of the vulnerabilities found in software are traced back to memory buffer overruns. Buffer overruns are conditions in which a pointer to a contiguously allocated block of memory goes past the bounds. Two major bugs in modern internet history are OpenSSL's Heartbleed and Cloudflare's Cloudbleed both being bugs in array bounds checking crept into software.
1. HeartBleed fix for OpenSSL - https://git.openssl.org/gitweb/?p=openssl.git;a=commitdiff;h=96db902 - This bug was introduced into OpenSSL in HeartBeat extensions for SSL connections keep-alive feature. Pointer was getting past without record length checks enforced in the fix commit diffs
2. CloudBleed fix for Cloudflare proxies HTML parsing - https://blog.cloudflare.com/incident-report-on-memory-leak-caused-by-cloudflare-parser-bug/. This is similar to HeartBleed buffer overrun, which happened during equality check for end of buffer pointer (which bypasses an increment check condition within Ragel HTML parser).

---

Duff's Device - 7 August 2018

---

Duff's device is C loop unrolling design pattern which mixes switch and looping for fall throughs. Example code is in code/duffs_device.c which decrements a counter in a loop - switch statement falls through when counter is not divisible by 5 to case statement selected by switch and number for chosen case statement is decremented by fall through to other cases till end. This is an optimized loop unrolling inspired by jmp statements with in loops in assembly language routines. Previous example prints (in "case: counter" format):

```
0: 29
1: 28
2: 27
3: 26
4: 25
0: 24
1: 23
2: 22
3: 21
4: 20
0: 19
1: 18
2: 17
3: 16
4: 15
0: 14
1: 13
2: 12
3: 11
4: 10
0: 9
1: 8
2: 7
3: 6
4: 5
```

```
0: 4
1: 3
2: 2
3: 1
4: 0
```

---

10 February 2020 - C++ Class simulation in C, Function pointers, Private and Public

---

Example code class_in_c.c simulates object oriented programming of C++ in C. C only has the facility of
struct {} declaration which is exploited to declare few private and public variables (variables are structs containing primitive type unions of type "struct variable". A flag to distinguish private variables is part of the struct variable) and member function pointers for class "struct classinc". Member private and public variables are assigned to by set<type>() functions which check private flag of a variable and deny access for private variables. Member function function1() clears the private flag and invokes set<type>() reinstating the private flag after for member variable thus simulating privileged member function access of private. set<type>() functions simulate type overloading. Logs in code/logs/class_in_c.log.10February2020 trace the public and private access by functions pointed to.

---

7 May 2020 - Simulation of loop by recursion, void typecast, Binary search tree example

---

Functional programming defines loop imperatives by recursion. Code example recursionasloop.c demonstrates
incrementing a counter by recursion without recourse to for loop. Function loop() defines two clauses
for simulating increment by recursion and to check if an array representation of a binary tree is indeed
a binary search tree by verifying if inorder traversal of the tree is sorted ascending and adheres to subtree comparison ordering.  Both int object and int[] array are passed to loop as void* where arg is reinterpret_cast()-ed.  Two binary trees one of which is not a BST are tested. Pairwise subtree comparison and loopless increment are printed to logs/recursionasloop.log.7May2020.

---

6,7,8 June 2020 - Dynamic Nested For Loop (variant of question from UGC NET/SLET/JRF)

---

Writing hardcoded nested for loops (of constant depth) in C is trivial. But for generalizing loops to
arbitrary nested depths is less straightforward because loop body has to be dynamically indented which
requires on-the-fly code generation. Code example dynamicnestedloop.c implements a contrived recursive
version of nested looping of arbitrary depth. Function nestedloop() accepts from,to and function pointer

to body of the loop. nestedloop() is recursively invoked along with loopbody() till depth is decremented
to 0. Subtle difference to usual n depth for loop of range 10 - counter is incremented to 10^n + 10^(n-1) + ... instead of 10^n (and thus powerful compared to hardcoded nested for loop) which is filtered by depth==1 check within loopbody(). Removing depth==1 clause increments counter to 1110 and not 1000 (depth 3 nesting). Iterative version of dynamic nesting is again non-trivial.

---

2,3,4,7 July 2020 - Arrays as Pointers, Recursive exponentiation in binary (variant of questions
22,48 from GATE 2020)

---

Questions 22 and 48 of GATE 2020 have been merged to code example multidimensionalarrays.c which
demonstrate:

- 2-d arrays as objects and their multidimensional pointer dereferencing
- exponentiation function pp(a,b) which raises a to the exponent b after converting b
  to binary string by modulo 2 division recursively (to-binary function - tob())

pp() function obtains the binary string of b from tob() which is looped through and tot is updated
only for bits "1" in binary string of b. Auto variable array arr[] is allocated on stack and dereferenced
by tob().