Day03.C

Introduction (3)
Numbers

Young W. Lim

December 9, 2017

0.1 positive decimal numbers

```
::::::::::::::
t1a.c
::::::::::::::#include <stdio.h>

int main(void) {

   printf("%4d   %4x\n",  0,  0);
   printf("%4d   %4x\n",  1,  1);
   printf("%4d   %4x\n",  2,  2);
   printf("%4d   %4x\n",  3,  3);
   printf("%4d   %4x\n",  4,  4);
   printf("%4d   %4x\n",  5,  5);
   printf("%4d   %4x\n",  6,  6);
   printf("%4d   %4x\n",  7,  7);
   printf("%4d   %4x\n",  8,  8);
   printf("%4d   %4x\n",  9,  9);
   printf("%4d   %4x\n", 10, 10);
   printf("%4d   %4x\n", 11, 11);
   printf("%4d   %4x\n", 12, 12);
   printf("%4d   %4x\n", 13, 13);
   printf("%4d   %4x\n", 14, 14);
   printf("%4d   %4x\n", 15, 15);
   printf("%4d   %4x\n", 16, 16);
   printf("%4d   %4x\n", 17, 17);
   printf("%4d   %4x\n", 18, 18);
   printf("%4d   %4x\n", 19, 19);
   printf("%4d   %4x\n", 20, 20);
   printf("%4d   %4x\n", 21, 21);
   printf("%4d   %4x\n", 22, 22);
   printf("%4d   %4x\n", 23, 23);
   printf("%4d   %4x\n", 24, 24);
   printf("%4d   %4x\n", 25, 25);
   printf("%4d   %4x\n", 26, 26);
   printf("%4d   %4x\n", 27, 27);
   printf("%4d   %4x\n", 28, 28);
   printf("%4d   %4x\n", 29, 29);
   printf("%4d   %4x\n", 30, 30);
   printf("%4d   %4x\n", 31, 31);

}

::::::::::::::
t1a.out
::::::::::::::
   0      0
   1      1
   2      2
   3      3
```

```
   4      4
   5      5
   6      6
   7      7
   8      8
   9      9
  10      a
  11      b
  12      c
  13      d
  14      e
  15      f
  16     10
  17     11
  18     12
  19     13
  20     14
  21     15
  22     16
  23     17
  24     18
  25     19
  26     1a
  27     1b
  28     1c
  29     1d
  30     1e
  31     1f
```

0.2 positive hexadecimal numbers

```
:::::::::::::::
t1b.c
:::::::::::::::
#include <stdio.h>

int main(void) {

   printf("%4d   %4x\n", 0x00, 0x00);
   printf("%4d   %4x\n", 0x01, 0x01);
   printf("%4d   %4x\n", 0x02, 0x02);
   printf("%4d   %4x\n", 0x03, 0x03);
   printf("%4d   %4x\n", 0x04, 0x04);
   printf("%4d   %4x\n", 0x05, 0x05);
   printf("%4d   %4x\n", 0x06, 0x06);
   printf("%4d   %4x\n", 0x07, 0x07);
   printf("%4d   %4x\n", 0x08, 0x08);
   printf("%4d   %4x\n", 0x09, 0x09);
```

```
   printf("%4d   %4x\n", 0x0a, 0x0a);
   printf("%4d   %4x\n", 0x0b, 0x0b);
   printf("%4d   %4x\n", 0x0c, 0x0c);
   printf("%4d   %4x\n", 0x0d, 0x0d);
   printf("%4d   %4x\n", 0x0e, 0x0e);
   printf("%4d   %4x\n", 0x0f, 0x0f);
   printf("%4d   %4x\n", 0x10, 0x10);
   printf("%4d   %4x\n", 0x11, 0x11);
   printf("%4d   %4x\n", 0x12, 0x12);
   printf("%4d   %4x\n", 0x13, 0x13);
   printf("%4d   %4x\n", 0x14, 0x14);
   printf("%4d   %4x\n", 0x15, 0x15);
   printf("%4d   %4x\n", 0x16, 0x16);
   printf("%4d   %4x\n", 0x17, 0x17);
   printf("%4d   %4x\n", 0x18, 0x18);
   printf("%4d   %4x\n", 0x19, 0x19);
   printf("%4d   %4x\n", 0x1a, 0x1a);
   printf("%4d   %4x\n", 0x1b, 0x1b);
   printf("%4d   %4x\n", 0x1c, 0x1c);
   printf("%4d   %4x\n", 0x1d, 0x1d);
   printf("%4d   %4x\n", 0x1e, 0x1e);
   printf("%4d   %4x\n", 0x1f, 0x1f);

}
::::::::::::::
t1b.out
::::::::::::::
   0      0
   1      1
   2      2
   3      3
   4      4
   5      5
   6      6
   7      7
   8      8
   9      9
  10      a
  11      b
  12      c
  13      d
  14      e
  15      f
  16     10
  17     11
  18     12
  19     13
  20     14
  21     15
  22     16
```

```
 23     17
 24     18
 25     19
 26     1a
 27     1b
 28     1c
 29     1d
 30     1e
 31     1f
```

0.3 negative decimal numbers

```
::::::::::::::\textbf
t1c.c
::::::::::::::
#include <stdio.h>

int main(void) {

   printf("%4d   %4x\n",  0,  0);
   printf("%4d   %4x\n", -1, -1);
   printf("%4d   %4x\n", -2, -2);
   printf("%4d   %4x\n", -3, -3);
   printf("%4d   %4x\n", -4, -4);
   printf("%4d   %4x\n", -5, -5);
   printf("%4d   %4x\n", -6, -6);
   printf("%4d   %4x\n", -7, -7);
   printf("%4d   %4x\n", -8, -8);
   printf("%4d   %4x\n", -9, -9);
   printf("%4d   %4x\n",-10,-10);
   printf("%4d   %4x\n",-11,-11);
   printf("%4d   %4x\n",-12,-12);
   printf("%4d   %4x\n",-13,-13);
   printf("%4d   %4x\n",-14,-14);
   printf("%4d   %4x\n",-15,-15);
   printf("%4d   %4x\n",-16,-16);
   printf("%4d   %4x\n",-17,-17);
   printf("%4d   %4x\n",-18,-18);
   printf("%4d   %4x\n",-19,-19);
   printf("%4d   %4x\n",-20,-20);
   printf("%4d   %4x\n",-21,-21);
   printf("%4d   %4x\n",-22,-22);
   printf("%4d   %4x\n",-23,-23);
   printf("%4d   %4x\n",-24,-24);
   printf("%4d   %4x\n",-25,-25);
   printf("%4d   %4x\n",-26,-26);
   printf("%4d   %4x\n",-27,-27);
   printf("%4d   %4x\n",-28,-28);
   printf("%4d   %4x\n",-29,-29);
```

```
   printf("%4d   %4x\n",-30,-30);
   printf("%4d   %4x\n",-31,-31);

}
::::::::::::::
t1c.out
::::::::::::::
   0      0
  -1   ffffffff
  -2   fffffffe
  -3   fffffffd
  -4   fffffffc
  -5   fffffffb
  -6   fffffffa
  -7   fffffff9
  -8   fffffff8
  -9   fffffff7
 -10   fffffff6
 -11   fffffff5
 -12   fffffff4
 -13   fffffff3
 -14   fffffff2
 -15   fffffff1
 -16   fffffff0
 -17   ffffffef
 -18   ffffffee
 -19   ffffffed
 -20   ffffffec
 -21   ffffffeb
 -22   ffffffea
 -23   ffffffe9
 -24   ffffffe8
 -25   ffffffe7
 -26   ffffffe6
 -27   ffffffe5
 -28   ffffffe4
 -29   ffffffe3
 -30   ffffffe2
 -31   ffffffe1
```

0.4 negative hexadecimal numbers

```
::::::::::::::
t1d.c
::::::::::::::
#include <stdio.h>

int main(void) {
```

```
   printf("%4d   %4x\n", -0x00, -0x00);
   printf("%4d   %4x\n", -0x01, -0x01);
   printf("%4d   %4x\n", -0x02, -0x02);
   printf("%4d   %4x\n", -0x03, -0x03);
   printf("%4d   %4x\n", -0x04, -0x04);
   printf("%4d   %4x\n", -0x05, -0x05);
   printf("%4d   %4x\n", -0x06, -0x06);
   printf("%4d   %4x\n", -0x07, -0x07);
   printf("%4d   %4x\n", -0x08, -0x08);
   printf("%4d   %4x\n", -0x09, -0x09);
   printf("%4d   %4x\n", -0x0a, -0x0a);
   printf("%4d   %4x\n", -0x0b, -0x0b);
   printf("%4d   %4x\n", -0x0c, -0x0c);
   printf("%4d   %4x\n", -0x0d, -0x0d);
   printf("%4d   %4x\n", -0x0e, -0x0e);
   printf("%4d   %4x\n", -0x0f, -0x0f);
   printf("%4d   %4x\n", -0x10, -0x10);
   printf("%4d   %4x\n", -0x11, -0x11);
   printf("%4d   %4x\n", -0x12, -0x12);
   printf("%4d   %4x\n", -0x13, -0x13);
   printf("%4d   %4x\n", -0x14, -0x14);
   printf("%4d   %4x\n", -0x15, -0x15);
   printf("%4d   %4x\n", -0x16, -0x16);
   printf("%4d   %4x\n", -0x17, -0x17);
   printf("%4d   %4x\n", -0x18, -0x18);
   printf("%4d   %4x\n", -0x19, -0x19);
   printf("%4d   %4x\n", -0x1a, -0x1a);
   printf("%4d   %4x\n", -0x1b, -0x1b);
   printf("%4d   %4x\n", -0x1c, -0x1c);
   printf("%4d   %4x\n", -0x1d, -0x1d);
   printf("%4d   %4x\n", -0x1e, -0x1e);
   printf("%4d   %4x\n", -0x1f, -0x1f);

}
::::::::::::::
t1d.out
::::::::::::::
   0      0
  -1   ffffffff
  -2   fffffffe
  -3   fffffffd
  -4   fffffffc
  -5   fffffffb
  -6   fffffffa
  -7   fffffff9
  -8   fffffff8
  -9   fffffff7
 -10   fffffff6
 -11   fffffff5
 -12   fffffff4
```

```
 -13   fffffff3
 -14   fffffff2
 -15   fffffff1
 -16   fffffff0
 -17   ffffffef
 -18   ffffffee
 -19   ffffffed
 -20   ffffffec
 -21   ffffffeb
 -22   ffffffea
 -23   ffffffe9
 -24   ffffffe8
 -25   ffffffe7
 -26   ffffffe6
 -27   ffffffe5
 -28   ffffffe4
 -29   ffffffe3
 -30   ffffffe2
 -31   ffffffe1
```

1 printing a variable's address and its content

```
::::::::::::::
t2.c
::::::::::::::
#include <stdio.h>

int main(void) {

   int a = 100;
   int *p = &a;

    printf("address(a)=%p \n", &a);
    printf("content(a)=%d \n",  a);

    printf("address(p)=%p \n", &p);
    printf("content(p)=%p \n",  p);

    printf("address(*p)=%p \n", &(*p));
    printf("content(*p)=%d \n",   *p);

}

::::::::::::::
t2.out
::::::::::::::
address(a)=0x7fff7581798c
content(a)=100
```

```
address(p)=0x7fff75817990
content(p)=0x7fff7581798c
address(*p)=0x7fff7581798c
content(*p)=100
```

- an integer variable a is defined
- its address is $0x7fff7581798c$
- the content stored at this addres is 100 in decimal
- the content is a data

- an integer *pointer* variable p is defined
- its address is $0x7fff75817990$
- the content stored at this address is $0x7fff7581798c$
- the content is the address of a, which is the value of p
- the address of variable $(*p)$ is $0x7fff7581798c$, which is the address of a
- the content stored at this is 100, which is the value of a

1.1 Size of Integer Types

```
::::::::::::::
t1.c
::::::::::::::
#include <stdio.h>


int main(void) {
  char  a1 = 0x10;
  short a2 = 0x2010;;
  int   a4 = 0x40304010;
  long  a8 = 0x8070605040302010L;

  printf("a1= %16x %20d\n", a1, a1);
  printf("a2= %16x %20d\n", a2, a2);
  printf("a4= %16x %20d\n", a4, a4);
  printf("a8= %16lx %20ld\n", a8, a8);

  printf("sizeof(a1)= %ld byte \n", sizeof(a1));
  printf("sizeof(a2)= %ld bytes\n", sizeof(a2));
  printf("sizeof(a4)= %ld bytes\n", sizeof(a4));
  printf("sizeof(a8)= %ld bytes\n", sizeof(a8));
}
```

```
::::::::::::::
t1.out
::::::::::::::
a1=                10                  16
a2=              2010                8208
a4=          40304010          1076903952
a8= 8070605040302010 -9191740941672636400
sizeof(a1)= 1 byte
sizeof(a2)= 2 bytes
sizeof(a4)= 4 bytes
sizeof(a8)= 8 bytes
```

size of integer types

- char : 1-byte integer type
- short : 2-byte integer type
- int : 4-byte integer type
- long : 8-byte integer type

%ld and %d

- to print 1, 2, 4-byte integer type in decimal, use %d.
- to print 8-byte integer type in decimal, use %ld (some computers may use %lld)

1.2 Bit Shift

```
::::::::::::::
t2.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  int i;
  int m;

  for (i=0; i<33; ++i) {
    m = (1 << i);
    printf("2^%2d    = %16ld\n", i, m);
  }

  printf("\nRAND_MAX= %16d\n", RAND_MAX);
}
```

```
::::::::::::::
t2.out
::::::::::::::
2^ 0    =                1
2^ 1    =                2
2^ 2    =                4
2^ 3    =                8
2^ 4    =               16
2^ 5    =               32
2^ 6    =               64
2^ 7    =              128
2^ 8    =              256
2^ 9    =              512
2^10    =             1024
2^11    =             2048
2^12    =             4096
2^13    =             8192
2^14    =            16384
2^15    =            32768
2^16    =            65536
2^17    =           131072
2^18    =           262144
2^19    =           524288
2^20    =          1048576
2^21    =          2097152
2^22    =          4194304
2^23    =          8388608
2^24    =         16777216
2^25    =         33554432
2^26    =         67108864
2^27    =        134217728
2^28    =        268435456
2^29    =        536870912
2^30    =       1073741824
2^31    =       2147483648
2^32    =                1

RAND_MAX=       2147483647

::::::::::::::
correct outputs
::::::::::::::
2^ 0    =                1
2^ 1    =                2
2^ 2    =                4
2^ 3    =                8
2^ 4    =               16
2^ 5    =               32
2^ 6    =               64
2^ 7    =              128
```

```
2^ 8    =              256
2^ 9    =              512
2^10    =             1024
2^11    =             2048
2^12    =             4096
2^13    =             8192
2^14    =            16384
2^15    =            32768
2^16    =            65536
2^17    =           131072
2^18    =           262144
2^19    =           524288
2^20    =          1048576
2^21    =          2097152
2^22    =          4194304
2^23    =          8388608
2^24    =         16777216
2^25    =         33554432
2^26    =         67108864
2^27    =        134217728
2^28    =        268435456
2^29    =        536870912
2^30    =       1073741824
2^31    =       2147483648
2^32    =       4294967296

RAND_MAX=       2147483647
```

powers of 2

- the default integer type is int (4-byte)
- 32 bits in 4-byte : maximum 31-bit shift is possible
- if i is larger than 31, then think (i%32)
- for 2^{32}, we need at least 33-bit - must use 64-bit (8-byte) integer
- the suffix L represent the constant number is in a 8-byte long type integer
- for the correct ouput, we must use long m and 1L << i
- $RAND\_MAX = 2^{31} - 1$ ($2^{15} - 1$ in some computers)

1.3 Random Number

```
::::::::::::::
t3.c
::::::::::::::
#include <stdio.h>
#include <stdlib.h>
```

```
void func(void) {
  int i, m;

  for (i=0; i<10; ++i) {
    m = rand() % 10;
    printf("%3d ", m);
  }

  printf("\n");
}




int main(void) {

  func();
  func();
  func();
  puts(" ");

  srand(1);
  func();
  srand(1);
  func();
  srand(1);
  func();
  puts(" ");

  srand(2);
  func();
  func();
  srand(2);
  func();
  func();

}

::::::::::::::
t3.out
::::::::::::::
  3   6   7   5   3   5   6   2   9   1
  2   7   0   9   3   6   0   6   2   6
  1   8   7   9   2   0   2   3   7   5

  3   6   7   5   3   5   6   2   9   1
  3   6   7   5   3   5   6   2   9   1
  3   6   7   5   3   5   6   2   9   1
```

```
  0   9   8   5   1   8   4   7   5   7
  2   9   9   3   7   9   4   3   7   0
  0   9   8   5   1   8   4   7   5   7
  2   9   9   3   7   9   4   3   7   0
```

int rand(void);

- https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm
- int rand(void);
- the C library function int rand(void) returns a pseudo-random number in the range of 0 to RAND_MAX.
- RAND_MAX is a constant whose default value may vary between implementations but it is granted to be at least 32767.

void srand(unsigned int seed);

- https://www.tutorialspoint.com/c_standard_library/c_function_srand.htm
- The C library function void srand(unsigned int seed) seeds the random number generator used by the function rand.
- seed This is an integer value to be used as seed by the pseudo-random number generator algorithm.

1.4 Unsigned and Singed Integers

```
::::::::::::::
t2.c
::::::::::::::
#include <stdio.h>

int main(void) {
  unsigned char u;
           char i;

  u = -1;
  i = -1;

  printf("u= %d %u \n", u, u);
  printf("i= %d %u \n", i, i);

  u = 255;
  i = 255;

  printf("u= %d %u \n", u, u);
  printf("i= %d %u \n", i, i);
```

```
  printf("(u>0)= %d \n", u>0); // (1)
  printf("(i>0)= %d \n", i>0);  // (2)

}

::::::::::::::
t2.out
::::::::::::::
u= 255 255
i= -1 4294967295
u= 255 255
i= -1 4294967295
(u>0)= 1
(i>0)= 0
```

Unsigned and singed 1-byte integer examples

- -1 in the 2's complement number system

1byte char	2 hexa digits	0xff	(-1)
2byte short	4 hexa digits	0xffff	(-1)
4byte int	8 hexa digits	0xffffffff	(-1)
8byte long	16 hexa digits	0xffffffffffffffff	(-1)

- 255 in the 2's complement number system

1byte char	2 hexa digits	0xff	(-1)
2byte short	4 hexa digits	0x00ff	(255)
4byte int	8 hexa digits	0x000000ff	(255)
8byte long	16 hexa digits	0x00000000000000ff	(255)

- unsigned char u;
  - u = 255 means u = 0xff
  - 0xff : 1111_1111
  - unsigned : not a 2's complement system, but all positive numbers
  - therefore 0xff is (255)
  - because it is a positive number, $(u > 0)$ evaluates true (1)
- (signed) char i;
  - i = 255 means i = 0xff
  - 0xff : 1111_1111
  - its 2's complement : 0000_0001 (+1)
  - therefore 0xff is (-1)
  - because it is a negative number, $(i > 0)$ evaluates false (0)