

Day03 A

Young W. Lim

2017-10-07 Sat

1 Based on

2 Introduction (3) - Numbers

- Numbers
- Memory

"C How to Program", Paul Deitel and Harvey Deitel

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

printf's argument

```
int printf(const char *format, ...);
```

- the function prototype
- declared in <stdio.h> file
- format : format string "a = %d \n"
- ... : variable number of arguments separated by commas

printf's variable number argument

- each %d expects a matching argument of int type
- two %d's in the format string
 - two integer arguments must be supplied
- `printf("---%d---%d---%d\n",`
 - 3 %d's — 3 integer arguments must be supplied
 - `printf("---%d---%d---%d\n", 10,20,30);`
 - `printf("---%d---%d---%d\n", a, b, c);`
where a, b, c are integer variables

printf's format specifiers

%d	signed decimal integer	to print 10 or -10
%x	unsigned hexadecimal integer	to print 10 or -10
%c	a single character	to print 'A' or 'Z'
%s	a string	to print "A" or "hi"

- `int i=10, j=20, k=30; // integer variable`
- `char c='A'; // character variable`
- `char *s="Hello"; // character pointer variable`
- `printf("i= %d in decimal\n", i);`
- `printf("j= %d in hexadecimal\n", j);`
- `printf("c= %c a single character\n", c);`
- `printf("s= %s a set of characters\n", s);`

the type int

- int : 4 byte memory space
- 1 byte = 8 bits
- 4 bytes = $4 * 8 = 32$ bits
- 2^{32} possible cases : 2^{32} numbers
 - **unsigned** int
all 2^{32} numbers are positive numbers : $[0 \sim 2^{32} - 1]$
 - **signed** int
two halves and each half has $2^{32} / 2 = 2^{31}$ numbers
 2^{31} positive numbers : $[0 \sim 2^{31} - 1]$
 2^{31} negative numbers : $[-2^{32} \sim -1]$
2's complement number system

1-byte number example

- consider the 8-bit (1-byte) integer 0x34
- prefix 0x : the following number
 - is not a decimal
 - but a hexadecimal
- $0x34 = 3 \cdot 16^1 + 4 \cdot 16^0 = 48 + 4 = 52$ in decimal
- finding $-0x34 = -52$ is the same as finding the 2's complement of $0x34 = 52$

2's complement of $0x34 = 52$

- for the 8-bit binary representation
- convert each hexadecimal digit into 4-bit binary number
3 --> 0011, 4 --> 0100
- form the 8-bit binary number $00110100 = 0x34 = 52$
- 2's complement of 00110100
 - flip each binary digit (0->1, 1->0)
11001011 (1's complement of $0x34$)
 - add +1 to this 1's complement
11001100 (2's complement of $0x34$)
- convert each 4-bit binary number into a hexadecimal digit
 $0xCC = -0x34 = -52$

4-byte number example (1)

- first, get the 32-bit (4-byte) binary representation
- $0x00000034$ (4-byte) = $0x34$ (1-byte) = 52
- form the 32-bit binary number
 $00000000\ 00000000\ 00000000\ 00110100 = 0x34 = 52$
- 2's complement
 - flip each binary digit (0→1, 1→0)
 $11111111\ 11111111\ 11111111\ 11001011$ (1's complement of $0x34$)
 - add +1 to this 1's complement
 $11111111\ 11111111\ 11111111\ 11001100$ (2's complement of $0x34$)
- convert each 4-bit binary number into a hexadecimal digit
 $0xFFFFFCC = -0x34 = -52$

verifying c code (1)

```
#include <stdio.h>

int main(void)
{
    int m = 0x34;

    printf("m= %08x hexadecimal \n", m);
    printf("m= %8d      decimal \n", m);
    printf("-----\n");
    printf("-m= %08x hexadecimal \n", -m);
    printf("-m= %8d      decimal \n", -m);
}
```

```
m= 00000034 hexadecimal
m=      52      decimal
-----
-m= ffffffffcc hexadecimal
-m=     -52      decimal
```

4-byte number example (2)

- first, get the 32-bit (4-byte) binary representation
- $0xFFFFFFFF$ (4-byte) = $0xFF$ (1-byte) = -1
- form the 32-bit binary number
 $11111111\ 11111111\ 11111111\ 11111111 = 0xFF = -1$
- 2's complement
 - flip each binary digit (0→1, 1→0)
 $00000000\ 00000000\ 00000000\ 00000000$ (1's complement of $0xFF$)
 - add +1 to this 1's complement
 $00000000\ 00000000\ 00000000\ 00000001$ (2's complement of $0xFF$)
- convert each 4-bit binary number into a hexadecimal digit
 $0x00000001 = -0xFF = +1$

verifying c code (2)

```
#include <stdio.h>

int main(void)
{
    int m = -1

    printf("m= %08x hexadecimal \n", m);
    printf("m= %8d      decimal \n", m);
    printf("-----\n");
    printf("-m= %08x hexadecimal \n", -m);
    printf("-m= %8d      decimal \n", -m);
}
```

```
m= ffffffff hexadecimal
m=      -1      decimal
-----
-m= 00000001 hexadecimal
-m=       1      decimal
```

Variables and Memory

- a variable
 - variable's address : a memory location
 - variable's value : the content of that location

address	content
a variable's address	a variable's value

Pointer variables and Memory

- `int a`
 - `a` : an integer variable
 - the address of `a` : `&a`
 - the value of `a` : `a` (an integer value)

- `int * p`
 - `p`: an integer pointer variable
 - the address of `p` : `&p`
 - the value of `p` : `p` (the address of an integer value location)
 - not an integer value
 - but an address of a memory location where an integer value is stored

* and & operators

- * Operator
 - followed by an address
 - either a variable address
 - or a constant address
 - returns the content of a memory location
- & Operator
 - followed by a variable
 - not followed by a constant
 - returns the address of that a variable's memory location