

CSE 351 Section 1 – Number Bases and Working in C [Solutions]

Hi there and welcome to section! ☺

Numerals

A *numeral* is a symbolic representation of a number. For the purposes of this class, we will define a numeral as a sequence of digits (symbols).

Number Bases

If we have an n -digit numeral $d_{n-1}d_{n-2} \dots d_0$ in base b , then the value of that numeral is $\sum_{i=0}^{n-1} d_i b^i$, which is just fancy notation to say that instead of a 10's or 100's place we have a b 's or b^2 's place.

The most common bases we will use in this class are 2, 10, and 16, which are called binary, decimal, and hexadecimal (or hex), respectively. In base b , each digit d_i can only be one of b fixed symbols (0-1 for binary, 0-9 for decimal, etc.).

The table on the right shows the equivalent numerals for the numbers 0 through 15 in these three major number bases. We differentiate between these bases by using the prefix '0b' for binary and '0x' for hexadecimal.

Binary	Decimal	Hex
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Exercises:

1. Complete the table below by converting the numbers into the other two common bases. You may leave the "Decimal" column unsimplified.

Binary	Decimal	Hexadecimal
0b10010011	$2^7 + 2^4 + 2^1 + 2^0 = 147$	0x93
0b10110	$1 \times 16^1 + 6 \times 16^0 = 22$	0x16
0b111111	63	0x3F
0b100100	$2^5 + 2^2 = 36$	0x24
0b110000110000	$12 \times 16^2 + 3 \times 16^1 = 3120$	0xC30
0b0	0	0x0
0b101110101101	$11 \times 16^2 + 10 \times 16^1 + 13 \times 16^0 = 2989$	0xBAD
0b110110101	437	0x1B5

Setting Up Your System

You have four options for your working environment:

- 1) CSE Labs: Log in locally a *Linux* machine in CSE1: 002, 003, 006 CSE2: 110, 124, 129 (need a CSE account)
- 2) Remote access: Log in remotely to `attu.cs.washington.edu` (CSE account)
- 3) Install the CSE VM: <https://www.cs.washington.edu/lab/software/linuxhomevm>
- 4) Personal computer: Must be running a Linux distribution (*e.g.*, Ubuntu, Fedora, CentOS)

You will need the following tools for the rest of the course, so make sure you know how to access/use them (already installed on `attu` and the VM) and start to get familiar with them:

- Text Editor (personal preference)
 - Try many, pick one! Some tutorials can be found on the course website.
 - Command-line: `nano`, `vim`, `emacs`
 - Graphical: `gedit`, `emacs`
- GNU Compiler Collection (`gcc`)
 - Example: `gcc -Wall -g -std=c18 -o execName sourceCode.c`
 - `-W` sets warnings
 - `-g` turns on debugging symbols
 - `-std` sets what version of C we are using
 - `-o` sets the name of the resulting executable
- GNU Project Debugger (`gdb`)
 - Command-line debugger that we will use heavily later in the course

Code Examples:

- 1) Download `HelloWorld.c` from the class webpage:

```
$ wget https://courses.cs.washington.edu/courses/cse351/22wi/sections/01/code/HelloWorld.c
```

- 2) Open the file in your favorite text editor and read the comments

- 3) Compile the file to the executable `hello`: `$ gcc -o hello HelloWorld.c`

- 4) Run the program: `$./hello`

- 5) Download `calculator.c` from the class webpage:

```
$ wget https://courses.cs.washington.edu/courses/cse351/22wi/sections/01/code/calculator.c
```

- 6) Read through the code in a text editor, then compile and run the program

- 7) Example usage: `$./calculator 4 5 +`

printf

Used to print to the console. Unfortunately, you can't concatenate String variables like you can in Java.

You provide a format string as the first argument, which includes placeholders to print out variables:

- `%d` for signed int, `%u` for unsigned int, `%f` for float, `%s` for "string", `%x` for hexadecimal, `%p` for pointer
- Examples:
 - `printf("I am %d years old", 20)` prints "I am 20 years old"
 - `printf("My name is %s", "Alfian")` prints "My name is Alfian"
 - `printf("%d in hex is %x", 2827, 2827)` prints "2827 in hex is 0xb0b"