USF Computer Science

CS 315 2017F

Search this site

Project01

Number Converter and Bit Twiddler Command Line Utility (nt)

Due Tuesday, September 12th 2016 at 11:59pm in your Github repository for Project01

Project01 will get you back into C programming and get you familiar with number representations that will be important to know well later in the course. You will implement a UNIX command line program, called nt (short for number tool), in C that can convert values between different number systems and also perform "bit twiddling." This little utility will prove useful for future projects.

Number Representation Conversion

Here is how it looks when you execute it with different values.

```
$ nt 713
0000 0000 0000 0000 0000 0010 1100 1001 (base 2)
0b00000000000000000000001011001001 (base 2)
0x000002C9 (base 16)
713 (base 10 unsigned)
713 (base 10 signed)
```

Note that nt detects that 713 is a integer number (more on this later). It then outputs the equivalent binary and the hexadecimal values. You need to output binary values in two forms: separate groups of 4 bits and a single number prefixed with 0b. The first format is easier to read, the second format is useful for cutting and pasting into a C program as constant literal. The default size for binary and hexadecimal is 32 bits (4 bytes).

```
$ nt -713
1111 1111 1111 1111 1111 1101 0011 0111 (base 2)
0b11111111111111111111110100110111 (base 2)
0xFFFFFD37 (base 16)
4294966583 (base 10 unsigned)
-713 (base 10 signed)
```

Note we get the binary and hexadecimal versions of the 2's complement representation of -713

```
$ nt 0b11100011
0000 0000 0000 0000 0000 0000 1110 0011 (base 2)
0b00000000000000000000000011100011 (base 2)
0x000000E3 (base 16)
227 (base 10 unsigned)
227 (base 10 signed)
```

In this case, nt recognized that 0b11100011 is a binary number and converted it to the equivalent unsigned base 10 equivalent of 227. The singed based 10 representation is also 227 because our default bit width is 32. It also displays the hexadecimal equivalent.

```
$ nt 0xE3
0000 0000 0000 0000 0000 0000 1110 0011 (base 2)
0b00000000000000000000011100011 (base 2)
```

```
0x000000E3 (base 16)
227 (base 10 unsigned)
227 (base 10 signed)
```

In this case nt recognizes 0xE3 as a hexadecimal number and converts it to binary, integer, and signed integer.

Changing the Bit Width

The default bit width for nt is 32. You can change this default with the -b option:

```
$ nt -b 16 713
0000 0010 1100 1001 (base 2)
0b0000001011001001 (base 2)
0x02C9 (base 16)
713 (base 10 unsigned)
713 (based 10 signed)

$ nt -b 16 -713
1111 1101 0011 0111 (base 2)
0b1111110100110111 (base 2)
0xFD37 (base 16)
64823 (base 10 unsigned)
-713 (base 10 signed)

$ nt -b 8 0xE0
1110 0000 (base 2)
0b11100000 (base 2)
0xE0 (base 16)
224 (base 10 unsigned)
-32 (base 10 signed)
```

You should support the following bit widths: 4, 8, 16, 32.

Bit Twiddling

You can select the range of bits to use for conversion using the -r option:

```
$ nt -b 4 -r 4,7 0b000010100000
1010 (base 2)
0b1010 (base 2)
0xA (base 16)
10 (base 10 unsigned)
-6 (base 10 signed)
```

Note that the first value in the range specifies the least significant bit (the rightmost bit) and the second value in the range specifies the most significant bit (the leftmost bit). The range values start at 0 and they are inclusive.

Testing

Your nt program should be able to support ANY 32 bit value in any representation (binary, hexadecimal, unsigned integer, signed integer). If the user provides a value that cannot fit into 32 bits then you should report an error:

```
$ nt 4294967297
Error: 4294967296 cannot fit into 32 bits.
```

You should test all the "corner" cases: 0, 1, 2, -1, -2, 2147483647, -2147483648, 4294967295, 4294967296. Also consider the largest numbers for the smaller bit widths.

Some requirements and notes:

- Your version of nt must implement the conversion algorithms described in Section 1.4 of DDCA where appropriate. In general you need to convert the string values (e.g.,

"-32", "0x1E", "0b1011") into integer (binary) form using the conversion algorithm. For output, you can use printf() as needed. You can use atoi() to parse the arguments to -b and -r.
- The argument given to nt will be a string (available in argv[1] or at a higher index if command line options are given).
- If no argument is given print an error message, then a usage message and exit.
- If an invalid number is provided, output and error message, then a usage message and exit.
- You can assume a maximum value of 32 bits and you should do error checking to ensure value that exceed 32 bits are rejected. If a different bit width is provided with -b you should ensure the value can fit in the specified number of bits.
- You need to get git working directly on your RPi.
- Must show how you can compile and run your solution on your RPi (no credit if you cannot do this)
    - If for some reason you have hardware problems with your RPi at the last minute you can ensure success by having your code in Github. You can clone your code to my RPi for demonstration purposes.

Extra Credit (Each worth one point)

- Submit 24 hours early (Monday September 11 at 11:59pm) and demo on Tuesday, September 12th.

Rubric

- (10 points) Clone GitHub repo onto RPi
- (10 points) Compile and run on RPi
- (5 points) Binary input
- (5 points) Hexadecimal input
- (5 points) Unsigned decimal input
- (5 points) Signed decimal input
- (5 points) Binary output
- (5 points) Hexadecimal output
- (5 points) Unsigned decimal output
- (5 points) Signed decimal output
- (5 points) -b bit width option
- (5 points) -r range option
- (5 points) Conforming output (your output must look my output exactly)
- (15 points) Simple tests and corner cases
- (10 points) Code quality: consistent formatting, consistent variable and function name formats, no extra vertical space, no commented out code, no redundant code, no extra long functions (including main), comments for tricky code.

Comments

You do not have permission to add comments.