

## Homework Assignment

---

### HW3: Integer/Hex Conversions

**Assigned:** 10 October 2019

**Due:** 31 October 2019 by class time

#### 1. Objectives

The objective of this assignment is to get you familiar with integer and character string manipulations. You are asked to convert an integer into its equivalent hex string, and then convert the hex string back to integer format. You are not allowed to use any integer to string manipulation library functions, such as itoa() etc. Make a subdirectory "hw3" in your cs240 directory for this assignment and copy the files from /courses/cs240/f19/xiaohui/GROUP/hw3. Use the gdb debugger to help you troubleshoot your program. Make a typescript that lists your programs, the execution results and intermediate debugging steps.

#### 2. Integer <-> Hex String Manipulation

(a) Write the C function itox() to convert an integer to a hex string. Its prototype is described in the header file xbits.h in this directory. Put your code in file xbits.c in the hw3 subdirectory of your cs240 directory. You can use the stub xbits.c from this directory to start with.

The idea behind the itox function is to convert an int variable (int is 32 bits on our machine) directly to an ascii string of hex numbers, '0','1'...'F'. In this exercise, we will only deal with **positive integers**.

The algorithm to convert decimal number to hex is to first divide the int number by 16, the remainder will form the first hex digit (the last character in the string). Use '0' to represent remainder = 0; '1' for remainder = 1; 'A' for remainder = 10; 'B' for remainder = 11 and so on. Next, divide the quotient by 16 and the remainder will form the second hex digit (next to the last). This process is repeated until the quotient is equal to 0 or all the characters in the string have been filled up. Then the algorithm stops. If the algorithm stops with quotient equal to 0, it fills the rest of the hex string with '0'. Print out the input integer number and its corresponding hex string to see if it is correct. Remember to terminate the string with '\0'.

Note that the array hexstring declared by the caller can be declared with a size  $2 * \text{sizeof}(\text{int}) + 1$ . Now sizeof() is a function evaluated at precompile time which gives the number of BYTES in a variable of a given type -- 4 bytes to an int on our machine, but it might be 8 bytes on a different machine and sizeof() would reflect that. There are 8 bits to a byte and so we expect  $2 * \text{sizeof}(\text{int})$  chars '0', '1',...'F' in a representation of the hex value. The +1 is to leave space for the final '\0' in the string.

(b) Write the C function xtoi() to convert a hex string to an integer. Its prototype is also described in xbits.h. Put your code in file xbits.c in the hw3 subdirectory of your cs240 directory. Your algorithm should only accept the valid hex digit 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

The algorithm to convert hex string to a decimal number is to start the string from the last character. Remember to skip over the '\0' character. Convert the corresponding character to its decimal equivalent (e.g. a character A will become 10) and multiply the number by powers of 16. The decimal equivalent of the last character will be multiplied by  $16^0$ ; the next by  $16^1$  and so on. The result is obtained by summing all the products together. Print out the hex string and its equivalent decimal number to see if it is correct.

## Homework Assignment

---

NOTE: The functions in `xbits.c` must be called by another program, and will not generate any output. So to test them you will have to write a driver (a main program that calls these functions and with appropriate arguments and outputs results -- this should be named `showxbits.c`). You must then compile the driver and link the compiled driver object code with the object code for the functions you write, combining them into a single executable. Note that there is a stub `showxbits.c` driver you can use to test the program and the code in `xbits.c` and `xbits.h`. To test bits initially, you can give the command to build and run the `showxbits` program:

```
gcc -m32 showxbits.c xbits.c -o showxbits
./showxbits
```

The `xbits.h` file is called a header file. The header file is the glue that makes sure `xbits.c` and the driver have the same prototypes for the functions in `xbits.c`. It guarantees that any correct driver can link with any correct implementation.

### 3. Interactive Integer <-> Hex String Manipulation

Rewrite `showxbits.c` so that it reads integers one at a time from `stdin` using the library function `scanf` (the equivalent to `printf` that does input -- look this up in K&R pg.157).

For each integer it reads it should write the input decimal, converted hexadecimal, and the reconverted decimal representations. The hexadecimal output strings requested by `showxbits` are to be calculated by calling `itox()`. To generate the reconverted decimal, you can call `atoi()`. Use `printf` with the `%x` format for integers input, `%s` format for hex strings you've created.

Use the value returned by `scanf` to end `showxbits` execution when it scans a non-integer (or EOF). When `scanf` is properly converting a single value under its conversion `%d`, it should return 1 (the number of tokens converted). See pg 245, `fscanf`, end of the first paragraph. The code you need to write in `showxbits` could look like this:

```
while (scanf("%d", &n) == 1) {
    itox( hexstring, n);
    m = atoi( hexstring);
    printf("%12d %s %12d\n", n, hexstring, m);
}
```

Note: `showxbits` program should *not* prompt the user for input. "No prompt" is a UNIX convention for reading files from `stdin`. There's a good reason for the convention. It allows you run `showxbits` easily and cleanly either by typing in the input data or using input redirection. Create a typescript to show your results.

### 4. Debugger gdb

Your typescript file should show that you have demonstrated the use of the `gdb` debugger. First build your program with the `gcc -g` option for `gdb`. Show the following `gdb` commands (shown in bold letters) in your typescript file:

## Homework Assignment

---

**gdb** progname

**b**(reak) main

**r**(un)

in some reasonable order:

**s**(tep) **n**(ext) (learn the difference!)

at any useful time, **p**(rint) some expression

**l**(ist) source code lines

**bt** see call stack

**help** for more information

**q**(uit) to conclude

\*\*\* NOTE: IF YOU SEE A RUNTIME ERROR THAT SAYS "core dumped", DELETE THE "core"  
FILE FROM YOUR SUBDIRECTORY RIGHT AWAY OR RISK RUNNING OUT OF DISK SPACE.  
\*\*\*\*\*