CS220 Spring 22 Exam 1 Practice - Solutions

1. Express **-77** as a 16-bit two's complement binary number.

   There are a couple of ways to do this one.

   <u>One way</u>

   The first power of two that is greater than -77 is -128. Solve the equation -128 + x = -77, so x = 51 so we have

   -128 + 32 + 16 + 2 + 1 in binary is 10110011. But the question asks for a 16 bit number, so you need to sign extend it to 1111111110110011.

   <u>Another way</u>

   Represent 77 in binary and then take the two's complement. 77 is 64 + 8 + 4 + 1 = 1001101. Now take the two complement by inverting the buts and adding one. But make sure we prepend with a 0 to make sure it isn't negative. So 77 is 01001101.

   ~01001101 = 10110010 and add 1 we get

```
10110010
       1
--------
10110011    Now sign-extend to 16 bits to get 1111111110110011.
```

2. What decimal number does the 8-bit two's complement number **0b11111111** represent?

   All ones is -1_{ten}. <u>To show work</u> you could negate the number by taking the two's complement, which is ~0b11111111 + 1 = 0b00000000 + 1 = 1. SInce the answer is 1 the original number must have been -1.

   Or you could have added up the powers of two, -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = -1.

3. What is the output of the following C code fragment.

`double pi = 3.14159;` `printf("%.2f\n", pi);`	**3.14**

4. What is the output of the program below? Answer: **<u>2147483647</u>**

```
printf("%d\n", 0x80000000 - 1);
```

You have to recall that the %d format specifier prints the 32 bit value in base 10. You also should have recalled

that 0x80000000 is the smallest possible 32 bit integer and subtracting one causes overflow and a wraparound to the largest positive integer 0x80000000 - 1 = 0x7FFFFFFF which is 2^{31} - 1 = 2147483647

5. What is the output of the statement below? **Answer: 4**

```
printf("%d", sizeof(int));  // four bytes in an int
```

6. What is the output of the statement below? **Answer: 1**

```
printf("%d", sizeof(char));  // one char is one byte
```

7. [3] What is the output of the statement below? **Answer: 1**

```
printf("%d", sizeof(unsigned char));  // unsigned does not change size
```

8. What will the following C code fragment print?

```
int x = 0xFFFFFFFE;
printf("%d\n", x + 1);  Answer: -1
```

Adding one to 0xFFFFFFFE is 0XFFFFFFFF, which is all ones on binary, which, we should know by now, is -1.

9. What is the smallest 32 bit two's complement integer?

   a. as a base ten integer: **-2147483648**

   b. in binary: **0b10000000000000000000000000000000**

   c. in hexadecimal: **0x80000000**

   d. as a power of two: **-2^{31}**

10. The output of the code below is **4**. For credit show values of **s** and **n** for each iteration of the while loop.

`int s = 0;`
`int n = 57;`
 	
`while (n > 0) {`
`  s = s + (n & 1);`
`  n = n >> 1;`
`}`
`printf("%d\n", s);`
	We have seen this code before. It just counts the numbers of ones in the binary representation of an integer. 57 = 32 + 16 + 8 + 1. So there are four powers of two (or ones in the binary number).

11. Fill in the body of the program below that prompts the user to enter an integer from the keyboard and the program should print the hexadecimal equivalent (recall the **x** format specifier for **printf**) as a C hexadecimal constant. See example output on right.

```
#include <stdio.h>
int main() {
    printf("Enter an integer: ");
    int n;
    scanf("%d", &n);
    printf("0x%X\n", n);
}
```

```
Enter an int: 44
0x2C
```

To get the constant printed out as a C hex constant you could have also used the # modifier on the x format specifier as in

```
printf("%#X\n", n);
```

12. Write a very short C code fragment that declares a variable **p** to be a pointer to an integer and have it point at the integer **x**. You can assume **x** is already declared.

```
int *p = &x;
```

13. [3] The code fragment below prints **Bar**

```
if (~0 != -1)           // ~0 is all ones, which is -1.
    printf("Foo");
else
    printf("Bar");
```

14. The **&** operator applied to a variable (as in) **&x** is called the **address-of** operator.

15. Assume we label the bits in a 32-bit integer left to right as $\mathbf{b_{31}b_{30}b_{29}...b_1b_0}$. Write a C program that reads a positive integer from the keyboard and prints the position of the leftmost one in the binary representation of the number. For example, if the user entered 33, the program would print 5, because 33_{ten} is **000...100001** and the 5th bit is a one.

```
int main() {
    int x;
    printf("Enter an integer: ");
    scanf("%ud",&x);
    int pos = -1;

    // Just keep counting until we hit zero.
    while (x > 0) {
        pos++;
        x = x >> 1;
    }

    printf("%d\n", pos);
}
```

16. Write a C function **reverse** that takes a string **s** as a parameter (an array of characters or a char *) and returns a new string that is the reverse of **s**. Use your function in a complete C program that reads words from the user (keyboard) and prints each word in reverse as in the sample output below. Put your program in a file named **exam1.c**. The **scanf** function returns **EOF** when the user types control-D and the **fgets** function returns **NULL** when the user types control-D. You can use either function to read a string from the user.

```
pi@raspberrypi-ehar:~/CS220 $ exam1
Word: hello
The reverse of hello is olleh
Word: python
The reverse of python is nohtyp
Word: applesauce
The reverse of applesauce is ecuaselppa
Word: <user typed control-D here>
```

MY SOLUTION ON NEXT PAGE. THERE A LOTS OF POSSIBILITIES.

```
/*
 * reverse a string s in a new string keeping the storage pointed
 * to by s unmodified.
 */
char * reverse(char *s) {

  // We need to calculate the length of s.
  int len = 0;
  while (s[len] != '\0')
    len++;

  // allocate storage for the reversed string.
  char *t = malloc(len+1);

  // copy characters from s to t
  int i = 0;
  while (i < len) {
     t[i] = s[len - i - 1];
     i++;
  }
  t[i] = '\0';

  return t;
}

// main.c
#include "reverse.h"
#include <stdio.h>  // scanf, printf
#include <stdlib.h> // malloc

int main() {
    const int MAX_WORD_LEN = 256;
    char buff[MAX_WORD_LEN];
    printf("Word: ");

    // could use fgets here instead of scanf
    while (scanf("%s", buff) != EOF) {
        printf("The reverse of %s is %s\n", buff, reverse(buff));
        printf("Word: ");
    }
}
```