Instructions
The exam is closed book, closed notes. You may not use a calculator, cell phone, etc.

For each of the questions of this quiz, you can assume the following sizes for C data types:

<table>
<thead>
<tr>
<th>type</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>1</td>
</tr>
<tr>
<td>short</td>
<td>2</td>
</tr>
<tr>
<td>int</td>
<td>4</td>
</tr>
<tr>
<td>long</td>
<td>8</td>
</tr>
<tr>
<td>float</td>
<td>4</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
</tr>
<tr>
<td>void*</td>
<td>4</td>
</tr>
</tbody>
</table>
For the following questions, you can assume that my home directory is the `jfiore` directory.

1. Unix shell stuff.

   (a) If I’m in my home directory i.e., `/home/jfiore`, what’s the single command that I can type in order to remove the assignment 2 files directory?
   
   **Solution:** `rmdir 2107/assign/02/files`

   (b) If I’m in my home directory, what’s the one command that I can type in order to create a `files` directory within the assignment 1 directory?
   
   **Solution:** `mkdir 2107/assign/01/files`

   (c) What can we type to see the file `stuff.txt` in hexadecimal along with an ASCII interpretation?
   
   **Solution:** `hexdump -C stuff.txt`

   (d) When I compile a file that contains some C functions but with no `main()` function, we compile it with the `-c` switch. Why? What specifically does the `-c` switch do?
   
   **Solution:** We’re running the compiler but not the linker.

   (e) If I’m in my home directory, what’s the one command that I can type to move `prog.c` to the `files` directory inside assignment 2?
   
   **Solution:** `mv prog.c 2107/assign/02/files`

   (f) If I run the command `gcc -E prog.c` to run the preprocessor only on `prog.c`, what does the resulting file contain (i.e., how is it different from `prog.c`)?
   
   **Solution:** Among other things, the `#include` statements are replaced with the contents of the files included, and the `#define` statements are processed.

   (g) If I’m in my home directory, what can I type to get a long directory listing (names, permissions, owner, length, etc) of the contents of the directory?
Solution: `ls -l`

(h) What’s the command that allows me to see what’s in `prog.c` one page at a time?

Solution: `more prog.c`

2. Some conversions.

(a) (1 point) 112 gbits = ? kbits

Solution: 112,000,000

(b) (1 point) 2 minutes = ? nanoseconds

Solution: \(2 \times 60 \times 10^9\)

(c) (1 point) 144 bits = ? mbytes

Solution: 0.00018 or \(144/8/10^6\)

(d) (1 point) 32 kbytes = ? bits

Solution: 256,000

(e) (1 point) 72 mbytes = ? tbits

Solution: 0.000576

Convert each of the following from base 10 to base 2 and base 16

(2 points) 3. 110

Solution:

\[
\begin{align*}
110 &= 55 \times 2 + 0 \\
55 &= 27 \times 2 + 1 \\
27 &= 13 \times 2 + 1 \\
13 &= 6 \times 2 + 1 \\
6 &= 3 \times 2 + 0 \\
3 &= 1 \times 2 + 1 \\
1 &= 0 \times 2 + 1 \\
\end{align*}
\]

\[110_{10} = 1101110_2 = 6E_{16}\]

(2 points) 4. 246

points: ______ out of a possible 14

2 of 9 exam continues...
Solution:

\[
246 = 123 \times 2 + 0 \\
123 = 61 \times 2 + 1 \\
61 = 30 \times 2 + 1 \\
30 = 15 \times 2 + 0 \\
15 = 7 \times 2 + 1 \\
7 = 3 \times 2 + 1 \\
3 = 1 \times 2 + 1 \\
1 = 0 \times 2 + 1 \\
\]

\[246_{10} = 11110110_{2} = F_{16}\]

5. Using the approximation trick that we talked about in class, about how much are each of the following?

(1 point) (a) \(2^{41}\)

(a) \(2\) trillion

(1 point) (b) \(2^{26}\)

(b) \(64\) million

(1 point) (c) \(2^{33}\)

(c) \(8\) billion

6. What is \(11110011_{2} + 1101110_{2}\) in base 2?

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\end{array}
\]

Which makes sense, because \(11110011_{2} = 243_{10}\), \(1101110_{2} = 110_{10}\) and \(243_{10} + 110_{10} = 353_{10}\), which is \(101100001_{2}\).

7. What is \(A53695_{16} + CF9B8_{16}\) in base 16?

\[
\begin{array}{cccccccc}
A & 5 & 3 & 6 & 9 & 5_{16} \\
C & F & B & 9 & 8_{16} \\
\hline
B & 2 & 3 & 2 & 2 & D_{16} \\
\end{array}
\]

In base 10, the same addition would be:

\[
\begin{array}{c}
10827413_{10} \\
+ \ 850840_{10} \\
\hline
11678253_{10} \\
\end{array}
\]

8. In hex, what is the smallest integer that can be represented by a 32-bit two's complement int?

8. \(0x80000000\)

points: ________ 3 of 9 exam continues... out of a possible 10
(1 point) 9. In hex, what is the largest integer that can be represented by a 32-bit two's complement int?

9. \texttt{0x7FFFFFFF}

(1 point) 10. In hex, what is the largest integer that can be represented by a 32-bit unsigned int?

10. \texttt{0xFFFFFFFF}

(1 point) 11. In hex, what is \(-1\) as a 32-bit two's complement int?

11. \texttt{0xFFFFFFFF}

(2 points) 12. What is printed by the following?

\begin{verbatim}
char x = 80, y=60;
char signed_sum;
unsigned char unsigned_sum;

signed_sum = x+y;
unsigned_sum = x+y;

printf("\%d\n", signed_sum);
printf("\%u\n", unsigned_sum); /* recall that \%u means to print as unsigned */
\end{verbatim}

\textbf{Solution:}

\begin{itemize}
  \item -116
  \item 140
\end{itemize}

13. \textbf{Some bit operations.} If we have \texttt{char i = 0x53, j = 0xF9;}, what is the result of the following operations? Your answer must be in the form of exactly two hex digits\(^1\).

\begin{verbatim}
\textbf{solution:} The easiest thing to do is to convert to binary first.

0x53_{16} = 01010011_2 \quad \text{and} \quad 0xF9_{16} = 11111001_2
\end{verbatim}

\begin{itemize}
  \item (1 point) (a) \texttt{i | j}

  \item (1 point) (b) \texttt{i & 0xFF}

  \item (1 point) (c) \texttt{\neg i}

  \item (1 point) (d) \texttt{i^j}

  \begin{itemize}
    \item (a) \texttt{0xfb}
    \item (b) \texttt{0x53}
    \item (c) \texttt{0x53}
    \item (d) \texttt{0xaa}
  \end{itemize}
\end{itemize}

\(\text{\textsuperscript{1}}\text{Ignore the possibility of promotion to 32-bit ints. Behave as though we're living in the land of 8-bit arithmetic.}\)

points: \underline{\hspace{3cm}} \hspace{2cm} 4 of 9 \hspace{2cm} question 13 continues...
(1 point) (e) !(i & 0)  
\[ (e) \quad 0x01 \]

(1 point) (f) i>>2  
\[ (f) \quad 0x14 \]

(1 point) (g) i && 0xFF  
\[ (g) \quad 0x01 \]

(6 points) 14. For this question, we’re doing 4-bit two’s complement representation of integers. Fill in the empty boxes in the following table. Addition and subtraction should be performed based on the rules for 4-bit, two’s complement arithmetic. Recall that in your book’s notation, **TMin** is defined to be the smallest negative two’s complement number that we can represent, and **TMax** is the largest positive one.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>n/a</td>
<td>−7</td>
<td>1001</td>
</tr>
<tr>
<td>n/a</td>
<td>−5</td>
<td>1011</td>
</tr>
<tr>
<td>n/a</td>
<td>−6</td>
<td>1010</td>
</tr>
<tr>
<td>Tmax</td>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>Tmin</td>
<td>−8</td>
<td>1000</td>
</tr>
<tr>
<td>Tmin+2</td>
<td>−6</td>
<td>1010</td>
</tr>
<tr>
<td>Tmax+2</td>
<td>−7</td>
<td>1001</td>
</tr>
</tbody>
</table>

15. fun with floats

(3 points) (a) How would we represent the number 199.5625\textsubscript{10} in fixed-point binary?

**Solution:**
- \[ 199_{10} = 11000111_2 \]
- \[ 0.5625_{10} = 0.1001_2 \]
- \[ 199.5625_{10} = 11000111.1001_2 \]

(1 point) (b) Normalize your answer from part (a).

**Solution:** \[ 11000111.1001_2 = 1.10001111001 \times 2^7 \]

(3 points) (c) How would 199.5625\textsubscript{10} be stored in a C 32-bit float variable? (Remember that for 32-bit floats, the bias value is 127.)

points: _______  
out of a possible 7
Solution:

sign  The number is positive, so the sign bit is 0.

exponent  We store the biased exponent. The bias is 127, so we have $7 + 127 = 134_{10} = 10000110_2$. So $10000110$ is stored in the exponent field.

mantissa  We had $1.10001111001_2 \times 2^7$, so in the mantissa field, we store $10001110001$ and then pad the remaining bits of the field with 0s. Remember that we have 32-bit floats. 1 bit was used for the sign, 8 bits were used for the exponent. This leaves $32 - (1 + 8) = 23$ bits for the mantissa. In the mantissa field, we store $100011110001000000000000$.

So we end up with:

<table>
<thead>
<tr>
<th>sign</th>
<th>exponent</th>
<th>mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10000110</td>
<td>100011110001000000000000</td>
</tr>
</tbody>
</table>

16. If I have the following:

```c
int main(void)
{
    int a=10;
    int b=20;

    int *p=&a;
    int *q=p;

    p++;
    (*q)++;  
}
```

and memory is laid out like this:

```
  a  1000
  b  1004
  p  1008
  q  1012
```

**Solution:** Before the increments, you end up with:

```
a  1000  10
b  1004  20
p  1008  1000
q  1012  1000
```

p++ increments the pointer p. (*q)++ increments what q points to. We get:

```
a  1000  11
b  1004  20
p  1008  1004
q  1012  1000
```

what do you see if you print:

(a) a

(1 point) (a) 11

points: ________ 6 of 9  question 16 continues...
(1 point) (b) &a

(b) 1000

(1 point) (c) b

(c) 20

(1 point) (d) &b

(d) 1004

(1 point) (e) p

(e) 1004

(1 point) (f) *p

(f) 20

(1 point) (g) &p

(g) 1008

17. Use the following code to answer the questions.

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int x;
    int *p;
    int A[4];
} junk;

void func(junk);

int main(void)
{
    int i=171;
    junk j;
    j.x=1;
    j.p=&i;
    *(j.p)=2;
    j.A[0]=3;
    func(j);
    return 0;
}

void func(junk j)
{
    j.x=1111;
    j.A[0]=3333;
    *(j.p)=37;
}
```

(a) (2 points) How many bytes are passed to the function func( )?

(a) 24

(2 points) (b) j.x

(b) 1

(2 points) (c) *(j.p)

(c) 37

(2 points) (d) j.A[0]

(d) 3

points: 7 of 9

out of a possible 8
(8 points) 18. Write a function which is passed an `int A[]` and its length. The function reverses the order of `A[]`. Do not use the `[ ]` operator.

**Solution:** One possibility:

```c
void rev_array(int A[], int len)
{
    int *b, *e;
    int t;
    b=(int*)A;
    e=b+(len-1);
    while (b<e) {
        t=*b;
        *b=*e;
        *e=t;
        b++;
        e--;
    }
}
```

(6 points) 19. Write a function `is_non_neg( )`, which takes as an argument an int `x`. The function returns 1 if `x >= 0`, or 0 if `x < 0`. Do not use the `<` or `>` operators. You cannot assume that ints are 4-byte values.

**Solution:** Remember the if the number is negative, the sign bit will be a 1.

```c
int is_non_neg(int x) {
    return !(x >> ((sizeof(int)*8)-1) & 0x01);
}
```

20. big-endian/little-endian

(5 points) (a) Write a function called `is_big_endian( )` which returns 1 if the machine running the function is big endian or 0 otherwise.

**Solution:**

```c
int is_big_endian()
{
    int x = 0xFF;
    char *p=(char*)&x;
    return !(*p&0xFF);
}
```

(1 point) (b) If we’re on a 32-bit little-endian machine and `int x` has the value 0x01234567, and x is stored starting at address 1000, what is the value of the byte stored at address 1002?
Solution: If it's little-endian, we'd have:

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0x67</td>
</tr>
<tr>
<td>1001</td>
<td>0x45</td>
</tr>
<tr>
<td>1002</td>
<td>0x23</td>
</tr>
<tr>
<td>1003</td>
<td>0x01</td>
</tr>
</tbody>
</table>

so, the answer is 0x23

(8 points) 21. Write a function `findstr()` which takes two arguments: a string n (think n for needle) and a string h (think h for haystack). The function returns the index of the first occurrence of n in h or -1 if n is not found. *Do not* use any function in `string.h`.

Solution:

(7 points) 22. Write a program which reads text from STDIN until EOF is entered. The program writes out all of the letters entered but with all of the consonants removed. Before quitting, the program prints the number of letters read and the number removed.

Solution:

During the exam, we said that it was ok to assume that if the letter read was not a vowel that it was a consonant.

```c
#include <stdio.h>

int main(void)
{
    int cur, /* current character */
        ccount=0, /* consonant count */
        lcount=0; /* all-letter count */

    while ((cur=getchar())!=EOF) {
        lcount++;
        if (is_vowel(cur))
            putchar(cur);
        else
            ccount++;
    }
    printf("Done.\n\n" );
    printf("There were %d letters total: %d consonants and %d vowels\n", lcount, ccount, lcount-ccount);

    return 0;
}
```