Tutorial 7: Float

Problem 1: Floating Point Representation

Consider the C code in Figure 1.

What are the outputs of the printf statements?

```c
#include <stdio.h>

int main (void) {
    static int a = 1;
    static float b = 1;
    int *c=&a;
    printf("Number \"1\" as integer = \"%x\"\n\n", *c);
    printf("Number \"1\" as pointed by an int pointer = \"%x\"\n\n", *(c+1));
    printf("Number \"1\" as float = \"%f\"\n\n", b);
    return 0;
}
```

Figure 1: Integer Vs Float

Integer (a = 1) is stored as (0x00000001) in memory. Float (b = 1) is stored as IEEE 754 format (0x3f800000) in memory. Printing the content of the memory locations where variables (a) and (b) are stored by the first two the printf statements will print (1) and (3f800000). The last printf statement will print the (b) as a decimal floating point value. In this program we rightly assume that (b) is located at a memory location immediately after (a). The way we access (b) by the pointer arithmetic probably should never happen outside the elec2041 class.

The printouts of the C program in Figure 1 are presented in Figure 2.

```
Number "1" as integer = "1"
Number "1" as pointed by an int pointer = "3f800000"
Number "1" as float = "1.000000"
```

Figure 2: The Outputs of printf Statements

Problem 2: Conversion from float to IEEE 754

Consider the C code in Figure 3.

What are the outputs of the printf statements?
#include <stdio.h>

int main (void)
{
    float a [] = {8.0, 8.5, 8.25, 8.125, 6.0, 6.5, 6.25, 6.125};
    int *c = a, i;
    for (i=0; i < 8; i++)
    {
        printf("Number ".3f" as IEEE 754 format = "%x", a[i], *(c+i));
    }
    return 0;
}

Figure 3: Float to IEEE 754 Format

The printouts from the C program in Figure 3 are presented in Figure 4. Program in Figure 3 prints the array (a []) elements as the float format in decimal fraction. It also prints its equivalent in IEEE 754 float format. The way it can print array (a []) in its equivalent form is by forcing an integer type pointer to point to a float type – an operation which is probably illegal outside the elec2041 class.

<table>
<thead>
<tr>
<th>Number</th>
<th>IEEE 754 Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.000</td>
<td>41000000</td>
</tr>
<tr>
<td>8.500</td>
<td>41080000</td>
</tr>
<tr>
<td>8.250</td>
<td>41040000</td>
</tr>
<tr>
<td>8.125</td>
<td>41020000</td>
</tr>
<tr>
<td>6.000</td>
<td>40c000000</td>
</tr>
<tr>
<td>6.500</td>
<td>40d000000</td>
</tr>
<tr>
<td>6.250</td>
<td>40c800000</td>
</tr>
<tr>
<td>6.125</td>
<td>40c400000</td>
</tr>
</tbody>
</table>

Figure 4: The Outputs of printf Statements

Problem 3: From IEEE 754 to Scientific Notation and Float

Consider the C code in Figure 5.
What are the outputs of the printf statements?

```
#include <stdio.h>

int main (void)
{
    int a [] = {0x3f000000, 0x388205ff, 0xb8324207, 0x3da8f5c3, 0x2cd31b32, 0xd4ae9f7c, 0x56a841ab, 0x5a1dbd91, 0x7fffffff, 0xffffffff, 0xff800000};
    float *c = a, i;
    for (i=0; i < 11; i++)
    {
        printf("IEEE 754 Representation \"%x\" is = \"%3.3e\", and = \"%3.3f\" \n\n", a[i], *(c+i), *(c+i));
    }
    return 0;
}
```

Figure 5: From IEEE 754 to Scientific Notation and Float
The printouts from the C program in Figure 7 are presented in Figure 8. Program in Figure 7 looks at the binary representation of a 32-quantity, interprets it as an IEEE 754 format representation, and breaks it into Sign, Exponent and Mantissa parts.

**Problem 4: From IEEE 754 to Binary Scientific Notation**

Consider the C code in Figure 7.

What are the outputs of the printf statements?

```c
#include <stdio.h>
int main (void)
{
    int a []= {0x3f000000, 0x388205ff, 0xb8324207, 0x3da8f5c3, 0x2cd31b32,
               0xd4ae9f7c, 0x56a841ab, 0xbf800000};
    int E, i;
    float M;
    char S;
    for (i=0; i < 8; i++)
    {
        S = (a[i] < 0) ? '-' : '+';
        E = ((a[i]&0x7fffffff) >> 23) - 0x7f;
        M = (a[i]&0x7fffff)/(8388608.0)+1;
        printf("IEEE 754 Representation "%x" = "+c%f X 2exp(%d)"

```
The printouts from the C program in Figure 9 are presented in Figure 10. In the first print statement (a) is so large that it consumes both (b) and (c). Therefore (a + b) and (a + c) are both evaluated as (a) and (d = a - a = 0).

The Float Number = "0.0000000e+00"

The Float Number = "1.0000000e-24"

Figure 10: The Outputs of printf Statements