CS 61C Spring 2022

Floating Point

Discussion 3

1 Pre-Check

This section is designed as a conceptual check for you to determine if you conceptually understand and have any misconceptions about this topic. Please answer true/false to the following questions, and include an explanation:

1.1 True or False. The goals of floating point are to have a large range of values, a low amount of precision, and real arithmetic results

1.2 True or False. The distance between floating point numbers increases as the absolute value of the numbers increase.

1.3 True or False. Floating Point addition is associative.

2 Memory Management

2.1 For each part, choose one or more of the following memory segments where the data could be located: **code, static, heap, stack**.

(a) Static variables

(b) Local variables

(c) Global variables

(d) Constants

(e) Machine Instructions

(f) Result of `malloc`

(g) String Literals

2.2 Write the code necessary to allocate memory on the heap in the following scenarios

(a) An array `arr` of k integers

(b) A string `str` containing p characters

(c) An $n \times m$ matrix `mat` of integers initialized to zero.

2.3 What's the main issue with the code snippet seen here? (Hint: `gets()` is a function that reads in user input and stores it in the array given in the argument.)

```
1 char* foo() {
2     char buffer[64];
3     gets(buffer);
```

```
4
5       char* important_stuff = (char*) malloc(11 * sizeof(char));
6
7       int i;
8       for (i = 0; i < 10; i++) important_stuff[i] = buffer[i];
9       important_stuff[i] = '\0';
10      return important_stuff;
11  }
```

Suppose we've defined a linked list **struct** as follows. Assume `*lst` points to the first element of the list, or is `NULL` if the list is empty.

```
struct ll_node {
    int first;
    struct ll_node* rest;
}
```

2.4 Implement `prepend`, which adds one new `value` to the front of the linked list. Hint: why use ll_node $**lst$ instead of ll_node$*lst$?

```
void prepend(struct ll_node** lst, int value)
```

2.5 Implement `free_ll`, which frees all the memory consumed by the linked list.

```
void free_ll(struct ll_node** lst)
```

3 Floating Point

The IEEE 754 standard defines a binary representation for floating point values using three fields.

- The *sign* determines the sign of the number (0 for positive, 1 for negative).
- The *exponent* is in **biased notation**. For instance, the bias is -127 which comes from -$(2^{8-1}-1)$ for single-precision floating point numbers.
- The *significand* or *mantissa* is akin to unsigned integers, but used to store a fraction instead of an integer.

The below table shows the bit breakdown for the single precision (32-bit) representation. The leftmost bit is the MSB and the rightmost bit is the LSB.

1	8	23
Sign	Exponent	Mantissa/Significand/Fraction

For normalized floats:
Value = $(-1)^{Sign} * 2^{Exp+Bias} * 1.\textbf{significand}_2$
For denormalized floats:
Value = $(-1)^{Sign} * 2^{Exp+Bias+1} * 0.\textbf{significand}_2$

Exponent	**Significand**	**Meaning**
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

Note that in the above table, our exponent has values from 0 to 255. When translating between binary and decimal floating point values, we must remember that there is a bias for the exponent.

3.1 Convert the following single-precision floating point numbers from binary to decimal or from decimal to binary. You may leave your answer as an expression.

- 0x00000000
- 8.25
- 0x00000F00
- 39.5625
- 0xFF94BEEF
- -∞
- 1/3

4 More Floating Point Representation

As we saw above, not every number can be represented perfectly using floating point. For this question, we will only look at positive numbers.

4.1 What is the next smallest number larger than 2 that can be represented completely?

4.2 What is the next smallest number larger than 4 that can be represented completely?

4.3 What is the largest odd number that we can represent? Hint: Try applying the step size technique covered in lecture.