CS 61C Summer 2016
Guerrilla Section 1: Number Representation & C

Question 0: *Warm Up*

1) Convert the following 8-bit two's complement numbers from hexadecimal to decimal:

0x80 = -128
0xF4 = -12
0x0E = 14

2) Assume that the most significant bit (MSB) of x is a 0. We store the result of flipping x's bits into y. Interpreted in the following number representations, how large is the magnitude of y relative to the magnitude of x? Circle ONE choice per row.

Unsigned	$\vert y\vert < \vert x\vert$	$\vert y\vert = \vert x\vert$	[✓ $\vert y\vert > \vert x\vert$]	Can't Tell
One's Complement	$\vert y\vert < \vert x\vert$	[✓ $\vert y\vert = \vert x\vert$]	$\vert y\vert > \vert x\vert$	Can't Tell
Two's Complement	$\vert y\vert < \vert x\vert$	$\vert y\vert = \vert x\vert$	[✓ $\vert y\vert > \vert x\vert$]	Can't Tell
Sign and Magnitude	$\vert y\vert < \vert x\vert$	$\vert y\vert = \vert x\vert$	$\vert y\vert > \vert x\vert$	[✓ Can't Tell]

· In unsigned, a number with the MSB of 1 is always greater than one with a MSB of 0.
· In one's complement, flipping all of the bits is the negation procedure, so the magnitude will be the same. ·
In two's complement, y is a negative number. Its magnitude can be found by applying the negation procedure, which is flipping the bits and then adding 1, resulting in a larger magnitude than x.
In sign and magnitude, the 2nd MSB bit will determine the relative magnitudes of x and y, so you can't tell for certain.

Question 1: *Silly Rabbit, Trits Are for Kids*

A new memory technology with three distinct states is exploding into the technology industry! Let's see if we can't develop some new number representations to take advantage of this new development.

(a) First, define a rule, analogous to what we use for binary numerals, for determining the unsigned value of a ternary numeral $d_n, d_{n-1} \ldots d_0$, and use this rule to convert 2102_3 into decimal:

unsigned($d_n, d_{n-1} \ldots d_0$) = $\sum_{i=0}^{n} 3^i \cdot d_i$

unsigned(2102_3) = 65

(b) Next we'd like to define an analogue to two's complement for ternary numerals, which we'll call three's complement. Three's complement numbers should be as evenly distributed between positive and negative as possible (favor negative if necessary), should have a zero at 0_3, and should increase in value when incremented as an unsigned value (except in the case of overflow). Define a rule for negating a three's complement number.

"flip" all of the trits (replace 2s with 0s, 0s with 2s), and add 1

(c) What is the most positive possible three's complement 8-trit number? Using this result, specify a rule for determining if a three's complement number is positive or negative.

$1 \ldots 1_3$. A number is negative if it has unsigned magnitude greater than $1 \ldots 1_3$

(d) There are two different two's complement numbers who are their own inverse. Specify these numbers.

0, 0b10…0

(f) What arithmetic operation is a shift left logical equivalent to with three's complement numbers?

Multiplication by a power of three

Question 2: *Wow! If only you could C the main memory (Fa 15, M1)*

Consider the following C program:

```
int a = 5;
void foo(){
    int temp;
}
int main()
{
    int b = 0;
    char* s1 = “cs61c”;
    char s2[] = “cs61c”;
    char* c = malloc(sizeof(char) * 100);
    foo();
    return 0;
};
```

1) Sort the following values from least to greatest: **&b, c, b, &temp, &a**.

b < &a < c < &temp < &b

2) For each of the following values, state the location in the memory layout where they are stored. Answer with *code*, *static*, *heap*, or *stack*.

s1	**stack**
s2	**stack**
s1[0]	**static**
s2[0]	**stack**
c[0]	**heap**

Question 3: *C Memory Model (Sp 15, M1)*

```
char *get_asdf_string_1() {
  char *a = "asdf";
  return a;
}
```

1) Does this function return a usable pointer to a string containing “asdf”? Yes
2) Which area of memory does the returned pointer point to? Static
3) Does this function leak memory? No

```
char *get_asdf_string_2() {
  char a[5];
  a[0] = 'a';
  a[1] = 's';
  a[2] = 'd';
  a[3] = 'f';
  a[4] = '\0';
  return a;
}
```

1) Does this function return a usable pointer to a string containing “asdf”? No
2) Which area of memory does the returned pointer point to? Stack
3) Does this function leak memory? No

```
char *get_asdf_string_3() {
  char *a = malloc(sizeof(char) * 5);
  a = "adsf";
  return a;
}
```

1) Does this function return a usable pointer to a string containing “asdf”? Yes
2) Which area of memory does the returned pointer point to? Static
3) Does this function leak memory? Yes

Question 4: *Linked Lists (Sp 15, M1)*

1) Fill out the declaration of a singly linked linked-list node below.

```
typedef struct node {
  int value;
  __struct node*__ next; // pointer to the next element
} sll_node;
```

2) Let's convert the linked list to an array. Fill in the missing code.

```
int * to_array(sll_node *sll, int size) {
  int i = 0;
  int *arr = __malloc(size * sizeof(int))___;
  while (sll) {
    arr[i] = ___sll->value___________________;
    sll    = ___sll->next____________________;
    _______i++_______________;
  }
  return arr;
}
```

3) Finally, complete `delete_even()` that will delete every second element of the list. For example, given the lists below:

Before: Node 1 → Node 2 → Node 3 → Node 4

After: Node 1 → Node 3

Calling `delete_even()` on the list labeled "`Before`" will change it into the list labeled "`After`". All list nodes were created via dynamic memory allocation.

```
void delete_even(sll_node *sll) {
  sll_node *temp;
  if (!sll || !sll->next) return;
  temp = ___sll->next____;
  sll->next = ___temp->next (or sll->next->next)_______;
  free(____temp_______);
  delete_even(____sll->next______);
}
```