Computer Science Fundamentals II
Problems from MIDTERM #1

Spring 2020

1. Below is a C++ program. **What does it output to the console?**

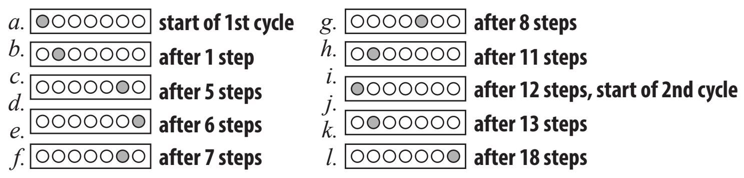
```
void add(int* x) {
  x[0] = x[0] + 4;
}

void swap(int p, int q) {
  int tmp = p;
  p = q;
  q = tmp;
}

int main(void) {
  int a = 10;
  int b = 20;
  int* c = &a;
  std::cout << a << " " << b << " " << c[0] << std::endl;
  c[0] = b;
  a = a + 1;
  std::cout << a << " " << b << " " << c[0] << std::endl;
  swap(a,b);
  std::cout << a << " " << b << " " << c[0] << std::endl;
  add(c);
  std::cout << a << " " << b << " " << c[0] << std::endl;
  int* d = c;
  c[0] = 1000;
  a = a * 2;
  std::cout << a << " " << b << " " << c[0] << " " << d[0] << std::endl;
}
```

2. An `int` array whose contents are 5, 4, 2, 0, 0, 2, 2, 4 has size 8 and its largest value is 5. It contains values from 0 to 5.

   Write a C++ function `counts` that takes three pieces of information: an array of integers `A`, the size of that array of integers `sz`, and the maximum value `max` held in that array of integers. You can assume that all the items in `A` are between 0 and `max`. Have the function build an array of integers on the heap that describes the number of occurrences of each value in `A`. It should then return that array of counts.

   For example, suppose `counts` is given an array with 8 values, the sequence 5, 4, 2, 0, 0, 2, 2, 4. Then `sz` would be 8 and `max` would be 5. It should give back an array with the sequence 2, 0, 3, 0, 2, 1 because the value 0 occurs two times, the value 2 occurs three times, the value four occurs twice, the value 5 occurs once, and the values 1 and 3 never occur.

3. A friend of yours is building a digital circuit with a display made up of a row of LED lights. When the display is operating, a single LED is lit up, and that pattern moves right and left. Below is a figure illustrating the behavior of their display with 7 LEDs. It starts with the leftmost LED lit up, as shown in *Figure a.*. We call this LED #0. After one step, the LED to its right is lit up instead. This is LED #1 and shown in *Figure b.* When the display reaches its 6th step, the far right LED #6 is lit up as shown in *Figure e.*. This behavior reverses direction so that after 7 steps LED #5 is lit up, as shown in *Figure f.*



You decide to model this LED display with C++ code. You invent a struct with this definition:

```
struct display {
  int numberLEDs;
  int stepsTaken;
};
```

This tracks the state of one of their LED display, one that has `numberLEDs` lights and has gone through a certain number of `stepsTaken`. **Write the C++ function**

`int whichIsLit(display d)`

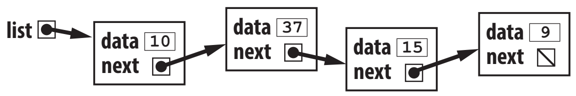
which, when given a `display` struct, returns which LED is lit up. If given a struct with `numberLEDs` set to 7 and `stepsTaken` equal to 0, it should return 0. If given a struct with `numberLEDs` set to 7 and `stepsTaken` equal to 7, it should return 5. You can assume that the number of LEDs in the display is at least 2.

Hint: The example display repeats every 12 steps when `numberLEDs` is 7. It would repeat every 2 steps when there are 2 LEDs. It repeats every 200 steps when there are 101. (NOTE: there was a mistake in my hint, fixed here. I was forgiving of code that relied on the incorrect hint.)

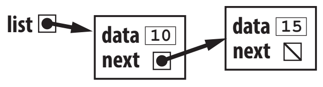
4. Below are definitions of two C++ structs to define a linked list data structure, one that stores a collection of integers.

```
struct node {
  int data;
  struct node* next;
};

struct llist {
  node* first;
}
```

Here is a picture of a `list` of type `(llist*)` storing the sequence 10, 37, 15, 9.



Write the C++ code for a function `void everyOther(llist * list)`. For linked lists of length 0 and length 1, it should do nothing. For lists of length 2 or more it should delete the second, fourth, sixth, etc. nodes in the list. It should keep the first, the third, the fifth, and so on. Here is a picture of the list above after `everyOther` was called on it.



5. **Write a C++ program** that asks for a positive integer and outputs an expression showing the powers of two that sum to that number. For example, the number 89 is a sum of 64, 16, 8, and 1. One can see that 64 is the largest power of two in 89 because it 128 is larger than 89. That sum can either be output as a decreasing sequence or as an increasing sequence, your choice. The decreasing version should behave like this:

```
$ ./powersum
Enter a positive integer: 89
89 = 64 + 16 + 8 + 1
$ ./powersum
Enter a positive integer: 12
12 = 8 + 4
$ ./powersum
Enter a positive integer: 8
8 = 8
```

   Here is a similar interaction, but with the powers of two reported in increasing order:

```
$ ./powersum
Enter a positive integer: 89
89 = 1 + 8 + 16 + 64
$ ./powersum
Enter a positive integer: 12
12 = 4 + 8
$ ./powersum
Enter a positive integer: 8
8 = 8
```

   You can either output as you go, printing the sum to `std::cout`. Or you can build a string (using, say, `std::to_string` to convert an integer to a string) and then output that string.