Oregon State University

CS 362 - Midterm, Question 2

The three cases I chose should cover most if not all the branches of this function. Case 1 is the case where n does not exist in container C. This case is really testing that the function does not delete anything in the container after calling removeAll with a value that doesn't exist in C. The only way to test this with just the provided signatures is to assert that size hasn't changed. Case 2 is that n exists some known k number of times in C. I will run this case with one negative value, one positive value, and zero to ensure the same behavior for negatives, positives, and zero. For all three values, I use both size and get to test correctness. The final case is when C is completely empty. This boundary case is testing whether the function does nothing and throws no error.

1. Test Case 1: n does not exist in C

   The below code will create a container that does not contain c. It will then copy that container, call removeAll, and then check that the size has not changed.

```
struct container C, testC;
&C = newContainer();
&testC = newContainer();

// populate C with values less than 100
for (int i = 0; i < 100; i++)
{
  add(i, &C);
}

// copy memory to testC
memcpy(&C, &testC, sizeof(struct container C));

// run function on testC
removeAll(101, &testC);

// check that size hasn't changed, i.e. nothing removed
assert(size(&C) == size(&testC));
```

2. Test Case 2: n exists in C k times

   The below code will create a container that contains n. It will then copy that container to another, call removeAll, and then check that the size has decreased by the number of times, k, that n was added to c. I will also called get(&C, n) and assert that it returns false. I will do this three times, one for positive k, one for negative k, and one for 0.

```
struct container C, blankC;
&C = newContainer();
&blankC = newContainer();
```

```
// populate C with values less than |100|
for (int i = 0; i < 100; i++)
{
  add(i, &C);
  add(-i, &C);
}

memcpy(&blankC, &C, sizeof(struct container));

for (int i = 0; i < 100; i++)
{
  add(300, &C);        // will check that number of instances of 300 ==
      100
}

removeAll(300, &C);
assert(size(&C) == 100);
assert(get(&C, 300) == 0);

memcpy(&C, &blankC, sizeof(struct container));

for (int i = 0; i < 100; i++)
{
  add(-300, &C);        // will check that number of instances of -300
     == 100
}

removeAll(-300, &C);
assert(size(&C) == 100);
assert(get(&C, 300) == 0);

memcpy(&C, &blankC, sizeof(struct container));

for (int i = 0; i < 100; i++)
{
  add(0, &C);        // will check that number of instances of -300 ==
     100
}

removeAll(0, &C);
assert(size(&C) == 100);
assert(get(&C, 0) == 0);
```

3. Test Case 3: C is completely empty

   The below code will create a container that has no values. Then it will call removeAll and check that size is 0 afterwards.

```
struct container C, testC;
&C = newContainer();

// run function on testC
removeAll(1, &C);

// check that size hasn't changed, i.e. nothing removed
assert(size(&testC) == 0);
```