CS302 -- Lab 2

- CS302 -- Data Structures and Algorithms II
- James S. Plank
- This file: **http://web.eecs.utk.edu/~jplank/plank/classes/cs302/Labs/Lab2**
- Lab Directory: **/home/jplank/cs302/Labs/Lab2**

What you are to submit

You are to submit the file `hashylvania.cpp` on Canvas. Do not submit anything else. That should contain all of your code, and should compile with no errors or warnings when you do:

```
g++ -Wall -Wextra -std=c++98 -o hashylvania hashylvania.cpp
```

The country of Hashylvania

The apocalypse has come, and along with a population of millions, you have been comported to the island of Hashylvania. As the political structure of the island is being formed, you have been appointed CTO, since you had a Raspberry Pi in your pocket when you were being comported. We can make up some back story about how we're powering the Pi, and how we had access to a monitor and keyboard, but you get the point.

Everyone on the island has a name and an ID. Both are strings without whitespace. An example would be:

```
Name: "Jim-Plank"  ID: "d6A8f81*bc6c3+efa8a_127ecY26667@"
```

There are no two people on the island with the same name and id.

The CMO (Chief mathmatical officer) is an old, bearded dude who fancies himself smart, but of course cannot program a computer. He has been tasked to derive an algorithm to randomly order the people. He has come up with the following algorithm:

- There is an official hash function of the island. I will show that to you below.
- When the time comes to order the people, he will choose a string (no spaces) and a modulus. The modulus is a number between 1 and 2^{64}-1.
- Each person will calculate his or her hash by concatenanting his or her name and id with the string, and hashing that.
- The ordering process goes as follows. We start with *h* equaling the hash of the random string. We then repeat the following:
  - Find all of the people whose hash modulo the modulus is closest *h* modulo the modulus. Ties are broken to favor higher values.
  - Now, of those people, find the person whose hash is the smallest hash greater than or equal to *h*. If there is no such person, then choose the person in that collection with the largest hash.
  - That is the next person in the order. Choose the next value of *h* by concatenating the person's id with the hexadecimal (lower case) representation of *h*, and hashing that.

You think that this algorithm is overly complex, and that the CMO should both shave and change his clothes every now and then. But that's not your place. You need to implement the algorithm.

Your Program

Your job is to write the program **hashylvania.cpp**. It will read strings from standard input. While the examples that I give you will be formatted, your input does not have to be -- it's just strings.

When you see the string "PERSON", you should read the next two strings -- that will be the person's name and id. Add them to the population.

When you see the string "ORDER", you should next read a string and a modulus (use **unsigned long long**). You should now order the people, and you should print the last person in the order, and then the first person in the order. Use the following format:

```
Last: name id
First: name id
```

Your error checking does not have to be robust. If you see an error, simply exit your program. You do not have to check for people who have duplicate names/ids. There is no limit on the sizes of the strings.

The "PERSON" and "ORDER" strings may arrive in any order. If you see "ORDER" and there are no people, simply ignore it and move on.

You may assume that when you're calculating the order, no two hashes are ever the same. If they are, you can blame the CMO.

The Hash Function

This is the official hash function of Hashylvania:

```
unsigned long long jsp_hash(const string &s)
{
  size_t i;
  unsigned long long h, t;

  h = 0x6bd300f29f1505ULL;

  for (i = 0; i < s.size(); i++) {
    t = h & 0x3ffffffffffffffULL;
    h = (t << 5) ^ h ^ s[i];
  }
  return h;
}
```

Exploring and Debugging

I am not going to walk you through examples. By this point in your career, you should be able to do that yourself. I will give you help, though. The program in the lab directory, **/home/jplank/cs302/Labs/Lab2/hashylvania** takes an optional command line argument (your program does not have to do this -- it's just there to help you). You may use it as follows:

- If the argument contains the character 'P', then whenever you see the string "ORDER", it will print the people, their hashes, and their hashes modulo the modulus:

```
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER C 10 | ./hashylvania P
Jim-Plank    H: 7050671006230931836 ->  6
Harvey-Plank H: 1267175115591939840 ->  0
Last: Harvey-Plank B
First: Jim-Plank A
UNIX>
```

- If the argument contains the character 'F', then whenever you see the string "ORDER", it will print each hash in the ordering, plus the (hash % modulus) values of the people it considering in each step:

```
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 10 | ./hashylvania FP
Jim-Plank    H: 6555409815530063263 ->  3
Harvey-Plank H: 2533495469764285539 ->  9
1st: 3397914012271970885  5 Closest:  3
1st: 3895987839514611439  9 Closest:  9
Last: Harvey-Plank B
First: Jim-Plank A
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 2 | ./hashylvania FP
Jim-Plank    H: 6555409815530063263 -> 1
Harvey-Plank H: 2533495469764285539 -> 1
1st: 3397914012271970885 1 Closest: 1
1st: 3895987839514611439 1 Closest: 1
Last: Harvey-Plank B
First: Jim-Plank A
UNIX>
```

- If the argument contains the character 'S', then whenever you see the string "ORDER", it will print each hash in the ordering, plus the hash of the person chosen next in the order:

```
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 10 | ./hashylvania FPS
Jim-Plank    H: 6555409815530063263 ->  3
Harvey-Plank H: 2533495469764285539 ->  9
1st: 3397914012271970885  5 Closest:  3
2nd: 3397914012271970885 Closest: 6555409815530063263 Jim-Plank
1st: 3895987839514611439  9 Closest:  9
2nd: 3895987839514611439 Closest: 2533495469764285539 Harvey-Plank
Last: Harvey-Plank B
First: Jim-Plank A
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 2 | ./hashylvania FPS
Jim-Plank    H: 6555409815530063263 -> 1
Harvey-Plank H: 2533495469764285539 -> 1
1st: 3397914012271970885 1 Closest: 1
2nd: 3397914012271970885 Closest: 6555409815530063263 Jim-Plank
1st: 3895987839514611439 1 Closest: 1
2nd: 3895987839514611439 Closest: 2533495469764285539 Harvey-Plank
Last: Harvey-Plank B
First: Jim-Plank A
UNIX>
```

- If the argument contains the character 'H', then whenever you see the string "ORDER", it will print each hash in the ordering, and how the next hash is calculated.

```
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 10 | ./hashylvania H
Old: 3397914012271970885 S: A2f27d0ca8ecb0a45 New: 3895987839514611439
Old: 3895987839514611439 S: B3611544c57828eef New: 6773836256035558274
Last: Harvey-Plank B
First: Jim-Plank A
UNIX> echo PERSON Jim-Plank A PERSON Harvey-Plank B ORDER ZZ 2 | ./hashylvania H
Old: 3397914012271970885 S: A2f27d0ca8ecb0a45 New: 3895987839514611439
Old: 3895987839514611439 S: B3611544c57828eef New: 6773836256035558274
Last: Harvey-Plank B
First: Jim-Plank A
UNIX>
```

Ok -- I lied. Here's an example walkthrough with the file **input.txt**:

- Part 1
- Part 2

Program structure and Running Time

Each "PERSON" statement should be *O(1)* and each "ORDER" statement should be *O(n log n)*, where *n* is the current number of people. My program structure was the following:

- I defined a **Person** to be a struct with strings **name** and **id**.
- I had a vector of pointers to people.
- When I saw "PERSON", I created a new person (with **new**) and added it to the vector.
- I defined a map **m** whose key is unsigned long longs, and whose val is another map.
- This other map has keys that are unsigned long longs and whose vals are pointers to people.
- When I saw "ORDER", I cleared out this map (which is actually unnecessary, but I did it anyway), and then I created it from the people, hash and modulus. For each person, I calculated its hash and (hash % modulus). I then made it so that there was an entry in **m** for (hash % modulus) and that the person was in the map for that entry, keyed by the person's hash.
- Then I went through the ordering process -- at each step, I'd find the person, delete them from their entry's map (and deleting their entry from **m** if they were the only person in their entry's map), and calculate the next hash. Along the way, I stored the first person in a (Person *).
- At the end, I printed the stored first person, and the last person.
- You have to delete the people and maps from **m**, or your program will not be *O(n log n)*.

Recommended Approach

Use my program and explore how the algorithm works using small numbers of people and small modulus values, like the examples above. Do not start coding until you understand your examples. It will be helpful for you to write a program that simply generates hashes from strings -- that way, you can verify that you are calculating the correct hashes.

Then program slowly, using my program to verify that yours is working correctly. Steps that you can take:

- Reading the people correctly.
- Calculating the hashes correctly (use 'H' and just one person).
- Inserting people into **m** and printing out **m**. My program doesn't do this, but you can verify youself that it looks right. Use two people and a modulus of 10 or 100. Then use two people and a modulus of 1.
- Find the first person and exit. Do this many times and verify with my program.
- Now delete the first person and print out the map before and after. Verify this by hand.
- Do the whole thing.