Goals

Implement a statically typed hash_map. See hash_map.h for full details

Submission instructions

This lab is due September 29th at 9pm

To submit this assignment you must upload TWO .cpp files to Brightspace. One of them is hash_map.cpp, which implements all the functions in hash_map.h, and the other is hash_list.cpp, which implements all the functions in hash_list.h. The version of hash_list.cpp that you upload MUST implement the iterator.

If you're working in a group then you both must submit these two files to Brightspace.

Restrictions

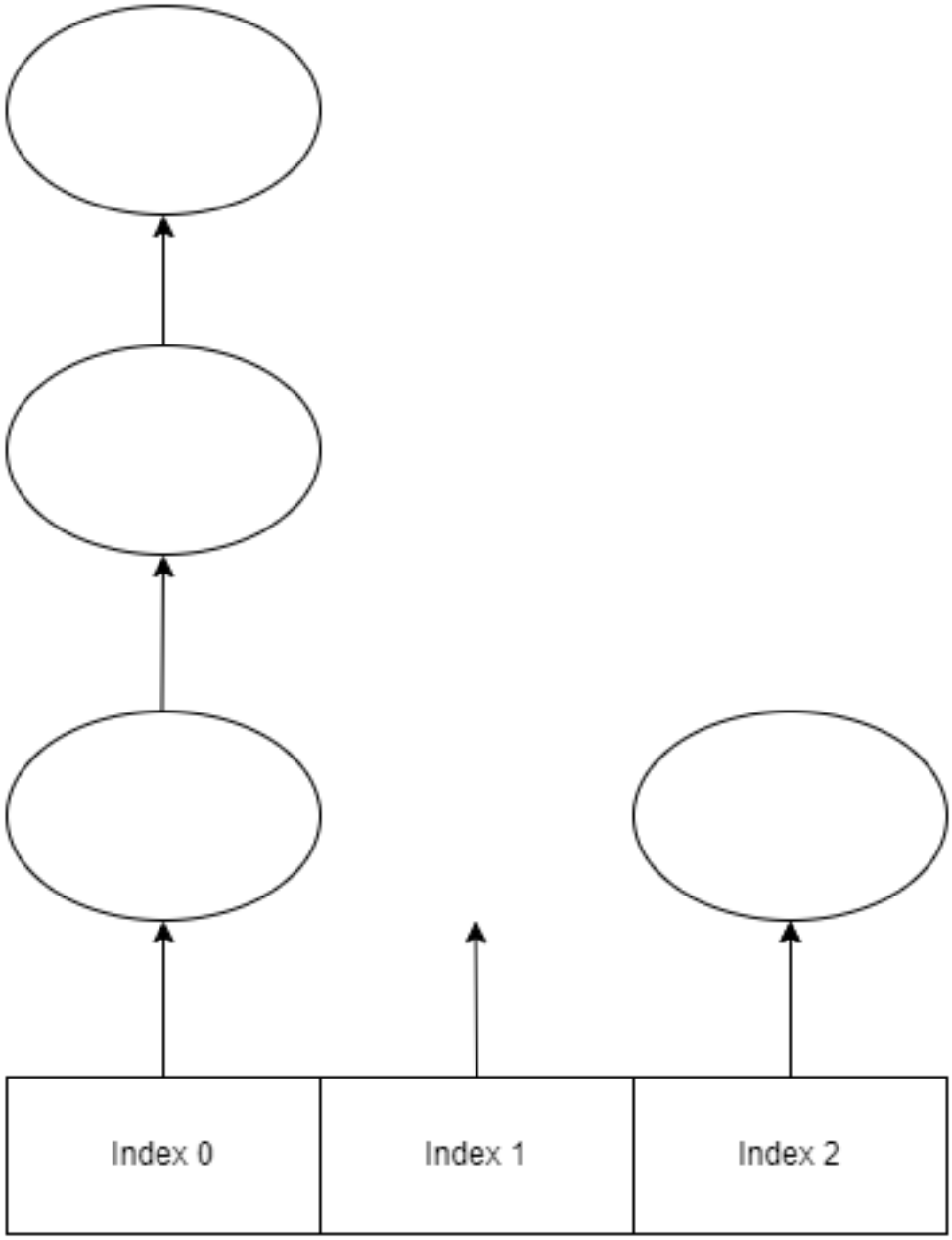
- You must submit a file called hash_map.cpp that implements all functions in hash_map.h and a file named hash_list.cpp that implements all of the functions declared in hash_list.h. The version of hash_list.cpp that you submit MUST include the iterator implementations.
- Your hash_map.cpp file is only allowed to include hash_map.h
- Your hash_list.cpp function is only allowed to include hash_list.h
- You may not use any standard containers.
- You aren't allowed to modify hash_map.h or hash_list.h in any way

Hashing function

In theory you could use any hash function you wanted, but we're requiring you to use the absolute value modulo _capacity (_capacity is defined in hash_map.h and is an argument to the constructor) as your hash function. If you don't use this hashing function then your get_bucket_sizes() implementation won't return the expected results and you won't get points for it.

get_bucket_sizes explanation

Let's imagine we have a map that has a capacity of 3 (this means the array of hash lists is 3 items long). We want to know how many elements are in each of those hash lists



The get_bucket_sizes function needs to populate the index of the passed in array with the size of the list at that index. So using the above example we get

```
list at index 0 has size 3
list at index 1 has size 0
list at index 2 has size 1
```

So we would set the passed array to

```
-------------------------
|   3   |   0   |   1   |
-------------------------
```

Compiling main.cpp

To compile your code using main.cpp call `make`. This generates an executable called `test` that you invoke using

```
./test
```

Running using our testing program

The testing program that we give you can be compiled using the provided make file by typing `make instructor_tests`. This will build an executable called test. You invoke test by calling

```
./test <grading file name> <trace file name>
```

Since your program needs to correctly run under valgrind the full command you should run to test your code is

```
valgrind --leak_check=full ./test grading_file.txt small_trace.txt
```

We recommend using the same trace files that we distributed for lab 1, although we also recommend you coming up with your own trace files. The format of the trace files for this lab is the same as in lab 1.