Homework #3

(3 points) Implement the `flat_hash_set` hash table you saw in lecture (see also `https://www.youtube.com/watch?v=ncHmEUmJZf4`). For the purposes of this assignment, it is sufficient to implement `insert` and `find` operations. The elements will be random 64 bit ints, you can use identity as hash function. You can also assume that the table has fixed size known beforehand (it is not necessary to implement table resizing).

Experiment: Insert N random values into a table of size M and measure the average search time for elements that: a) are present in the table, b) are not in the table (i.e. successful vs. unsuccessful search). Compare your implementation to `std::unordered_set`. How efficient is your hash table in terms of the load factor N/M (50–95%)?

Hints and notes:

- For SSE2 instructions such as `_mm_cmpeq_epi8`, you need to `#include <emmintrin.h>` (see `https://intel.ly/2nTEFye`).
- Position of the last `1` bit can be found in `gcc` via `__builtin_ctz` or `__builtin_ffs` functions (see `https://bit.ly/32oX19x`); the last `1` bit can be cleared using the Kernighan trick: `x &= x - 1;` (see `https://stanford.io/33JeVnl`, `https://stanford.io/2VUASNU`).
- Test your program properly (e.g. against `std::unordered_set`) – efficiency is important but correctness is more important!
- The easiest way to do random unsuccessful searches is to e.g. insert only odd values and then search for even values.