This lab investigates finding cache parameters and measuring peak values for “Memory Bandwidth” (in megabytes per second), (primarily) the influence of the caches (L1 and L2), and (secondarily) the influence of the number of different pages being accessed (in most modern systems memory is organized into pages of 4-8K bytes per page).

1. Get copies of the sample program `cache_info.c` and the current version of the `Makefile`. It may be difficult to find cache parameters for some systems but on Linux it is easy. You just have to read various files below the subdirectory 
   `/sys/devices/system/cpu/cpu0/cache/index[012]/`
   This program does not require the timing routines so you can just compile and link it with 
   ```
   gcc -g cache_info.c -o cache_info
   ```
   Run it to find out all of the L1 and L2 cache parameters for `sleipnir`.

2. Get copies of the sample program `cache.c` and the current version of the `Makefile`. Note that `cache.c` needs the auxiliary timing routines `timing.c` and `timing.h`. Compile and link the program.

3. The program reserves a large array in which you will write selected values, skipping by a fixed number of entries each time. Specifically, if you enter 127 for `line_offset` and `tmparray[j]` has just been written, the next entry to be written will be `tmparray[j + 127]`. Although for two close line offsets (e.g. 4093 and 4096) very nearly the same entries in the array are accessed, and very nearly the same locality with regard to page usage is observed there is a definite difference in profiling times if all other factors are equal. In practice, a persistent problem with timing memory intensive programs is variability of the results due to other running jobs which compete for cache and paging. It is best to perform a sequence of runs using a small interactive script, for example
   ```
   while true
     do
       echo "127" | ./cache | grep Bandwidth
       sleep 10
     done
   ```
   The peak value will usually become clear after about half a dozen to a dozen runs through the loop. You will have to run the program more than once and eliminate any outliers in the data. The last run will always be recorded in the logfile `cache_0127.log`.

4. Do this with `line_offset` equal to the each following values:
   ```
   127
   251
   509
   1021
   2039
   4093
   256
   512
   1024
   2048
   4096
   ```
   Note that the first column of numbers above are all prime numbers close to a power of 2 which follows in the second column. For each of the numbers you should record a peak value for “Memory Bandwidth,” so, when you are done you will have a table of values.

   **Questions** Summary your results for peak memory bandwidth in a table and include them as part or your writeup as an ascii text file with the answers to the following questions:
1. The general trend for the peak Memory Bandwidth values as one moves down either column 1 or column 2 is decreasing. How is this related to the range of values which are being accessed in the array?

2. The peak Memory Bandwidth values in column 1 (i.e. for prime values) are larger than the peak Memory Bandwidth values in column 2 (i.e. for powers of 2) even though the locality and the total number of pages accessed is practically the same. Why?