Introduction

Consider a factory with 30 machines M_i ($1 \le i \le 30$) making hot dogs and another 25 machines P_j ($1 \le j \le 25$) packing hot dogs. The hot dogs will be first made by the *making machines* and then sent to a *hot dog pool* [Note 1]. The *packing machines* will take hot dogs from the pool for packing. The *manager* of the factory specifies the number of hot dogs need to be done, *N* [Note 2]. There is a *log file* containing records for logging when a hot dog is packed. When all the hot dogs are done, the manager will append a summary to the log file.

Tasks

Using ***Pthreads*** and ***mutex***, design and implement a multithreaded C++ program under Linux that works as follows:

- Each making machine is modeled with a making thread (created by the master thread).
  - A making machine starts with checking the remaining number of hot dogs to be made.
  - The making machine makes one hot dog at a time.
  - The making machine takes 4 to 6 seconds [Note 3] to make one hot dog.
  - After a hot dog is made, the making machine takes 1 to 2 seconds to send the hot dog together with its machine id, *i*, to the hot dog pool.
  - Once the making machine finishes sending the hot dog, it can start to make the next hot dog.
  - When the required number of hot dogs is made, the making machine stops.
- Each packing machine is modeled with a packing thread (created by the master thread).
  - A packing machine starts with checking if there is any hot dog in the pool.
  - If there are hot dogs in the pool, the packing machine takes 1 to 2 seconds to take a hot dog from the pool, one at a time.
  - After taking a hot dog from the pool, the packing machine takes 1 to 2 seconds to pack the hot dog.
  - After packing the hot dog, the packing machine writes a record to the log file in form of < *hot_dog_id*, *i*, *j* >, where *hot_dog_id* is a serial number of the packed hot dog (counting from 1 to *N*), *i* is the making machine id and *j* is the packing machine id.
  - The packing machine takes 1 to 2 seconds to write a record to the log file.
  - Once the packing machine finishes writing the record, it can start to pack the next hot dog.
  - When all the hot dogs are packed, the packing machine stops.
- A master thread is created to model the manager of the factory.
  - The manager specifies the number of hot dogs to be done, *N*.
  - The manager initiates 30 making and 25 packing machines with machine id *i* and *j* respectively, where $1 \le i \le 30$ and $1 \le j \le 25$.
  - The manager has to wait until all machines stop, i.e., all hot dogs are made and packed and all records are written in the log file.
  - After all machines stop, the manager reads the log file and appends the following summary to the log file:
    - The number of hot dogs made and packed by each machine, in form of *<machine type (M or P), machine id (i or j), number of hot dogs made or packed >*
    - The total number of hot dogs made and packed, in form of *<total number of hot dogs made, total number of hot dogs packed>*

Notes:

1. There is NO specific insertion/removal order for the hot dog pool. If you need to enforce a certain order, specify this assumption clearly as a comment in your program.
2. The number of hot dogs need to be done should be an input value to your program.
3. To simulate the time taken for a thread to finish a step, you can make the thread sleep for a random period of time. For example, `sleep(rand()%3+4)` can be used to simulate the time taken for a making machine to make a hot dog, ranging from 4 to 6 seconds (http://linux.die.net/man/3/sleep, http://www.tutorialspoint.com/cplusplus/cpp_numbers.htm).

Program requirements and marking scheme

- Design and use of multithreading and mutex (**60%**)
  - Complete, correct and thread-safe multithreaded design and implementation including
    - thread management
    - mutual exclusion
    - a good balance between correctness and concurrency
  - Non-multithreaded implementation without mutex (0%)
- Program correctness (**30%**)
  - Complete and correct implementation of other features including
    - correct logic and coding of (thread and non-thread) functions
    - program input and output conform to the format of the sample output (refer to the attached sample output file)
    - successful program termination
  - You program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03). **If an executable file cannot be generated and running successfully on our Linux servers, it will be considered as unsuccessful.**
- Programming style and documentation (**10%**)
  - Good programming style
  - Clear in-line comments in the program file to state any assumptions made and describe the program design and logic (no need to submit a separate file for documentation)
  - Program completed in one file
  - Unreadable program without any comment (0%)

Submission

- This assignment has to be done individually or by a group of two students. You are encouraged to discuss the high-level design of your solution with your classmates but you **must** implement the program on your own. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of two files: a source program file (.cpp file) and a text file (.txt file) containing the log generated by your program **when *N* is set to 200**.
- Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp, 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp, 5xxxxxxx_5yyyyyyyy.txt for group submission. Only ONE submission is required for each group.
- Submit the files to Canvas.
- The deadline is **10:00 a.m.**, **6-APR-18 (Friday)**. No late submission will be accepted.

Questions?

- Contact our TA Mr. LIU Dawei at daweiliu2-c@my.cityu.edu.hk or the course lecturer.

Flowchart (for reference only)