**HW 3 Pseudocode**

Reminders:

1. Use POSIX thread library and compile with -pthread with the following command: gcc -Wall -Werror hw3-main.c hw3.c -pthread
2. Disable buffered output to stdout with the following command: setvbuf( stdout, NULL, \_IONBF, 0 );

Goals:

1. Parse command line arguments
   1. First argument: integer m = # of rows
   2. Second argument: integer n = # of columns
   3. Third argument: integer r for row of starting square (may range from 0 to m – 1)
   4. Fourth argument: integer c for column of starting square (may range from 0 to n – 1)
   5. Validation:
      1. Both m and n must be integers greater than 2
      2. Both r and c must be within valid ranges
      3. If invalid, display the message below and return EXIT\_FAILURE:

ERROR: Invalid argument(s)

USAGE: a.out <m> <n> <r> <c>

1. Create a synchronized multi-threaded solution to the knight’s tour problem 🡪 whenever more than one next possible move from a given position is possible, generate n threads to account for those n next possible moves
   1. Deterministic thread creation order (if Sonny’s in row a and column b, start looking for moves at row a – 2 and column b – 1 checking for moves counter-clockwise from there)
2. Update global variables as necessary
   1. If a knight’s tour is accomplished, increment variable total\_tours
   2. If a dead end is encountered per the restrictions, update the variable max\_squares if the number of squares covered by the leaf node thread exceeds the value in max\_squares (note that max\_squares will equal m x n if a knight’s tour is possible where m and n represent the row and column dimensions respectively)
   3. Variable next\_thread\_id starts at 1 🡪 for each new thread, increment next\_thread\_id and use the incremented value for that thread’s ID
3. Required output:
   1. Multiple possible moves detected
   2. Dead end reached
   3. Full knight’s tour completed
   4. Variable max\_squares updated
   5. Variable total\_tours updated
   6. Child thread joined back in to its parent thread
   7. Example:

bash$ ./a.out 3 3 0 0

MAIN: Solving Sonny's knight's tour problem for a 3x3 board

MAIN: Sonny starts at row 0 and column 0 (move #1)

MAIN: 2 possible moves after move #1; creating 2 child threads...

T1: Dead end at move #8; updated max\_squares

T2: Dead end at move #8

MAIN: T1 joined

MAIN: T2 joined

MAIN: Search complete; best solution(s) visited 8 squares out of 9

* 1. Full knight’s tour example output:

bash$ ./a.out 3 4 0 0

MAIN: Solving Sonny's knight's tour problem for a 3x4 board

MAIN: Sonny starts at row 0 and column 0 (move #1)

...

T5: Sonny found a full knight's tour; incremented total\_tours

...

MAIN: Search complete; found 2 possible paths to achieving a full knight's tour

* 1. Note that output interleaving of middle lines is expected (first few lines and last line must match, however) and assigned thread IDs will vary

1. NO\_PARALLEL mode
   1. Use debug mode 🡪 compile with the following command: gcc -Wall -Werror -D NO\_PARALLEL hw3-main.c hw3.c -pthread
   2. Debug mode example:

#ifdef DEBUG\_MODE

printf( "the value of q is %d\n", q );

printf( "here12\n" );

printf( "why is my program crashing here?!\n" );

printf( "aaaaaaaaaaaaagggggggghhhh!\n" );

#endif

* 1. Compiling in debug mode without NO\_PARALLEL flag: gcc -Wall -Werror -g -D DEBUG\_MODE hw3-main.c hw3.c -pthread

1. Error handling for threads
   1. In general, if an error is encountered in any thread, display a meaningful error message on stderr by using either perror() or fprintf(), then abort further thread execution by calling pthread\_exit(). Only use perror() if the given library function or system call sets the global errno variable.

Error messages must be one line only and use the following format:

ERROR: <error-text-here>

Restrictions and allowances:

1. Moves must be valid legal knight moves
2. The knight may not land on the same square twice
3. The knight’s starting and ending square may be different (a cycle is not necessary)
4. (0, 0) depicts the board’s upper left corner – the row and column numbers are command line arguments
5. Use calloc to dynamically allocate memory for the m x n board (calloc for array of m pointers and for each of these pointers use calloc again for array of size n) and use free to ensure program is memory leak/error free (do not use malloc or realloc) – make sure to allocate and free memory for each child thread you create