# COP 4520 Programming Assignment 4
# BONUS ASSIGNMENT
# CDSChecker Assignment:
# Checking Correctness of a Lock-Free Linked List

Point of contact: Christina Peterson, Email: clp8199@knights.ucf.edu

**Abstract**

Use the CDSChecker tool to test the correctness of the lock-free linked list presented in Chapter 9.8 of *The Art of Multiprocessor Programming* by Maurice Herlihy and Nir Shavit. The lock-free linked list must be implemented with the atomic instructions from the C++ Atomic Operations Library.

## I. Submission Guidelines

Please, submit your work via Webcourses.
Submissions by e-mail will not be accepted.
Due date: Monday, April 11th by 11:59 PM
Late submissions are not accepted.

## II. Grading

Grading of this assignment will be based on the following criteria:

- Documentation, including the experience report and CDSChecker log files (50/100pt).
- Test files, including the linked list implementation and unit test adapted for CDSChecker (50/100pt).
- You must use atomic instructions from the C++ Atomic Operations Library for the lock-free linked list implementation to receive full credit for the assignment.

## III. Deliverables

An archive containing:

- A report describing your experience with the tool. The report should include:
  - An interpretation of the CDSChecker statistics and bug report.
  - A written description of the changes that were made to resolve the bugs.
- A CDSChecker log file showing the errors that were found.
  - CDSChecker prints the statistics and bug report to the screen, so the log file may consist of a screen shot of the information.
- A CDSChecker log file showing that the errors were fixed.
- The linked list source code files, submitted in the same manner as the previous assignment.
- An archive containing files necessary for testing of your container.

## IV. Setting Up CDSChecker

CDSChecker can be download and installed by following the information described on project's site: plrg.eecs.uci.edu/?page_id=42. For your convenience we present the command line representation of these instructions in Figure 1.

```
git clone git://demsky.eecs.uci.edu/model-checker.git
cd model-checker
make
./run.sh test/userprog.o
```

Fig. 1. Setting up the project

## V. Working with CDSChecker

Setting up your testing environment can be done as follows:

- Create a linked list test directory in the model-checker directory
- Copy the Makefile from the 'model-checker/test' directory into your linked list test directory
- Modify the Makefile as necessary
  - Remove lines 8 and 9.
    - DIR := litmus include
    - $(DIR)/Makefile
  - Add `-std=c++11` to CPPFLAGS.
- NOTE: If you prefer to use your own Makefile, please be aware of the following issues:
  - You must edit your Makefile as follows:
    - BASE := path to model-checker
    - LIB_NAME:= model
    - INC:= -I$(BASE) -I$(BASE)/include
    - LIB:= -L$(BASE) -l$(LIB_NAME)
  - Make all object files using $(INC) and $(LIB)
- Create a header file within the linked list test directory and place the entire linked list implementation into the header file. If the linked list implementation consists of a .h file and .cc, .cpp, .cc, etc., be sure to combine all sources into a single header file. Otherwise, compilation will result in undefined references to user main.
  - The header file must include the `<atomic>` library.
- Create a main.cc file following the specifications described in the 'Running your own code' section on the project's website: plrg.eecs.uci.edu/?page_id=42. For your convenience, the guidance for using `<threads.h>` is listed below:
  - Include the CDSChecker threads library `<threads.h>`
  - Instead of using the standard main method `main(int argc, char *argv[])`, you must use the CDSChecker main method `user_main(int argc, char *argv[])`
  - Declare a thread using the data type `thrd_t` (example: `thrd_t` is equivalent to `pthread_t`)
  - Spawn a thread using `thrd_create` (example: `thrd_create` is equivalent to `pthread_create`). Note the following signature: `thrd_create(&t, (thrd_start_t) &work, (void *) arg)`
    - `t` = reference to thread `t` of type `thrd_t`
    - `work` = the code that the thread will run
    - `arg` = the argument that is passed to the thread
  - Wait for a thread to finish using `thrd_join` (example: `thrd_join` is equivalent to `pthread_join`). Note the following signature: `thrd_join(t)`
    - `t` = reference to thread `t` of type `thrd_t`
  - Define the thread code using `static void work(void *obj){ /*code to be performed by thread */ }`
    - CDSChecker is designed for unit tests rather than an entire program.
  - Create a unit test.
    - Create a linked list object within `user_main`.
    - Pass the address of the linked list object as an argument to the threads.
    - Define a thread for each of the supported operations of the linked list.
    - Each thread will invoke the linked list method once. Example:
      - `static void add(void *obj) { MyList *list = (MyList *) obj; list.add(); }`
      - `static void remove(void *obj) { MyList *list = (MyList *) obj; list.remove(); }`
      - `static void contains(void *obj) { MyList *list = (MyList *) obj; list.contains(); }`
    - Compile the unit test using 'make'.
    - Navigate to the model-checker directory. Run your program using CDSChecker as follows:
      - ./run.sh 'linked list test directory'/main.o
    - The CDSChecker statistics are printed to the screen. If any bugs are detected, they will be described in a bug report.
    - Prepare the written documentation according to the submission requirements described in Section III.