18-642 Recitation #9

March 30, 2018
Updates

• Homework:
  – Homeworks #29, #30 due Sunday 4/1 @ 11pm
  – Please take a look at recitation exercises before recitation

• Project:
  – Project 7 due tonight @ 11pm
  – Project 8 out, due 4/6 @ 11pm
  – Project 9 will be due 4/13, Project 10 will be due 4/27

• Be sure to fill out question sheet to receive attendance points today!
Today

- Project 8
- Race condition brainstorming
Project 8

- Compiler warnings
  - Run, but not have to fix
- Unit testing
  - Study example given
  - Write your own
Compiler Warning Flags

• Updated makefile (CMakeLists.txt)

```cpp
# Warning flags for Projects 8,9,10

target_compile_options(ece642rtle_student PUBLIC
"-Werror" # Do not comment this out after Project 8!
"-Wextra"
"-Wall"
"-Wfloat-equal"
"-Wconversion"
"-Wparentheses"
"-pedantic"
"-Wunused-parameter"
"-Wunused-variable"
"-Wreturn-type"
"-Wunused-function"
"-Wredundant-decls"
"-Wreturn-type"
"-Wredundant-decls"
"-Wunused-value"
"-Wswitch-default"
"-Wunused-iniitialized" "-O1" "-Winit-self"
}
```
Compilation Notes

- Makefiles only rebuild source files if they have changed (or if they haven’t successfully compiled before)
- This means:
  - The first time you build a project, you will probably see warnings
  - If the file compiles error-free and you don’t change the file, it won’t be included in the second compilation
  - You won’t see the warnings in the second compile
- Our solution: keep the –Werror flag
  - Turns all warnings into errors
  - Compilation won’t succeed until all warnings are removed
- If you disable the –Werror flag, remember to re-enable it and check build for warnings (important for Project 9)
Test your statechart

- Test all transitions
- Instrument a unit test framework to provide input variables
  - If you’re familiar with Java mock objects or C++ boost templates, this will be familiar
- For Project 8, you do not have to fix all failing unit tests
Example

```cpp
move_state moveTurtle(move_state curr_state, bool at_end) {
    switch (curr_state) {
        case MOVE_FORWARD: // S1
            // Outputs
            setOrientation(UP);
            if (at_end) { // T1
                curr_state = MOVE_FORWARD;
            } else if (will_bump()) { // T2
                curr_state = MOVE_BACK;
            } else { // Default: no transition
                curr_state = MOVE_FORWARD;
            }
            break;
        case MOVE_BACK: // S2
            // Outputs
            setOrientation(DOWN);
            if (at_end) { // T3
                curr_state = MOVE_BACK;
            } else if (will_bump()) { // T4
                curr_state = MOVE_FORWARD;
            } else { // Default: no transition
                curr_state = MOVE_BACK;
            }
            break;
        default:
            ROS_ERROR("Reached default state");
            break;
    }
}
```

- Can write tests that provide `curr_state` and `at_end` and test resultant state return value just by calling `moveTurtle()` with different values
- What’s missing?
Example

Need to test outputs and any inputs received by calling student_maze methods.
Create mock objects

- Change where student_turtle gets its header file based on whether testing is defined
  
  ```c
  #define testing 1
  #ifdef testing
  #include "student_mock.h"
  #endif
  #ifndef testing
  #include "student.h"
  #include "ros/ros.h"
  #endif
  
  move_state moveTurtle(move_stat
  ```

- Write methods that allow you to arbitrarily set inputs

- Might have to mock up things like array access (get_visit_index(i,j)) instead of visits[i][j])

```c
static orientation mock_orientation;
static bool mock_bump;
static bool mock_error = false;

/* Functions called by dummy_turtle */
void setOrientation(orientation ornt) {
  mock_orientation = ornt;
}

bool will_bump() {
  return mock_bump;
}

/* Functions used to instrument CUnit tests */
orientation test_orientation_result() {
  return mock_orientation;
}

void mock_set_bump(bool bump) {
  mock_bump = bump;
}

/* Dummy ROS_ERROR */
void ROS_ERROR(std::string e) {
  mock_error = true;
  std::cout << e << std::endl;
```
Write unit tests

- Test T2:
  - Set input (will_bump == true) using mocks
- Call state chart code with starting state of transition (S1) and any other inputs (at_end == false)
- Check outputs (in starting state, output is UP)

```c
void test_t2() {
  mock_set_bump(true);
  move_state return_state = moveTurtle(MOVE_FORWARD, false);
  orientation output_orientation = test_orientation_result();

  CU_ASSERT_EQUAL(output_orientation, UP);
  CU_ASSERT_EQUAL(return_state, MOVE_BACK);
}
```
Write unit tests

- Test T2:
  - Set input (will_bump == true) using mocks
- Call state chart code with starting state of transition (S1) and any other inputs (at_end == false)
- Check outputs (in starting state, output is UP)
  - Wait... why does the test check outputs of the starting state?
Write unit tests

• Test T2:
  – Set input (will_bump == true) using mocks
• Call state chart code with starting state of transition (S1) and any other inputs (at_end == false)
• Check outputs (in starting state, output is UP)
  – Wait... why does the test check outputs of the starting state?
  – Because we are testing what happens in the S1 block up to and as a result of the transition.

```c
case MOVE_FORWARD: // S1
  // Outputs
  setOrientation(UP);

if (at_end) { // T1
  curr_state = MOVE_FORWARD;
} else if (will_bump()) { // T2
  curr_state = MOVE_BACK;
} else { // Default: no transition
  curr_state = MOVE_FORWARD;
}
break;
```
Coverage

• 100% transition coverage
  – Test every transition

• Near 100% data coverage
  – All combinations of inputs, not just the ones that are supposed to trigger transitions
  – For example, at_end == false and will_bump == false
  – This might get intractable, so you only have to write up to 8 additional tests

• 100% branch coverage
  – Transition coverage should get you mostly there
  – Be sure to test all branches of all subroutines that the state machine code calls
Plan your time!

• Slack built in to Project 8
  – Don’t have to fix warnings
  – Don’t have to fix all failing unit tests

• Carries over to Project 9 (week-long project)
  – Fix all warnings
  – Fix all unit tests
  – AND write an invariant
Race condition brainstorming

• Give an example of a race condition or other concurrency problems that you have either encountered in person or are inherently present in systems you've personally used.

• If you don't have an idea that hasn't been proposed before, ask for help from the audience for a concurrency problem, and then suggest a solution on your own.