Umeå University
Institution för Datavetenskap

Datavetenskapens byggstenar 7.5 p
DV160HT15

OU1 Testing

Submitted 2015-11-17
Author: Lorenz Gerber (dv15lgr@cs.umu.se)
Instructor: Lena Kallin Westin / Johan Eliasson

Contents

1 Introduction

The aim of this laboration was to write unit test code for an implementation of the data type ‘queue’. The unit tests shall return feedback to the user, indicating ‘success’ or ‘failure’. The interface of the datatype may not be changed. All operations given in [1, p. 155] shall be tested.

2 Material and Methods

The datatype ‘queue’ was said to be implemented according to the specifications given in [1, pp.155 – 172]. The implementation was in ‘C’ according to standard ‘C99’. I chose to implement the unit tests according to the axiomatic specification given in [1, pp.156 + 157]. The toolchain was clang, CMake 3.3.2, GDB 7.8, IDE was CLion 1.2.1. Valgrind was used to find memory leaks. Development platform was OSX 10.10.

3 Results

3.1 General

The source code was stored in a file `queuetest.c` and submitted according instructions on the ‘cambro’ web interface. The unit tests were all implemented as `void` functions taking no arguments. All function output went to `stdout`. Each function outputs a short description of the tested operation. Depending on success, ‘pass’ or ‘fail’ is written out. In case of ‘fail’, the function exits and returns the value `1`. All test functions are called from `main.c`.

3.2 Unit Tests - Formal Descriptions

1. Axiom 1, use `empty` to create an empty list and check with `isEmpty` if it is really emtpy.
2. Axiom 2, apply `enqueue` to an empty list and check that `isEmpty` returns `FALSE`.
3. Axiom 3, if a queue q is empty, it follows that consecutive `enqueue`, `dequeue` will result in the same queue q.
4. Axiom 4, if a queue q is not empty, it follows that `dequeue` and `enqueue` follow commutative properties, hence the resulting queue q will look the same independent in which sequence `dequeue` and `enqueue` are applied.
5. Axiom 5, if a queue q is empty, sequential application of `enqueue(v, q)` and `front(q)` will return `v`.
6. Axiom 6, if a queue q is not empty, `enqueue` will not affect the next `front` operation.

4 Discussion

4.1 Unit Tests - Interpretation

1. Test 1: Fails if `empty` does not work. If `isEmpty` was hard coded to `TRUE`, this test could still be passed.

2. Test 2: If this test passes, we know that `isEmpty` works, hence we know also that `enqueue` can add a value to the queue. If it fails, either `isEmtpy` was hardcoded or `enqueue` did not add a value to the queue.

3. Test 3: If test 3 fails, there is something wrong with `dequeue`. If test 3 passes, we know that both `enqueue` and `dequeue` can add respectively remove a value from the queue. But we don't know anything whether they act on the correct end of the queue.

4. Test 4: If it fails, we know that either `enqueue` or `dequeue` act on the wrong end of the queue. However, if both act on the wront end, the test will still pass.

5. Test 5: when it fails, `front` can not read correctly a value from the queue. If it passes, we still don't know if it reads from the correct end.

6. Test 6: If it passes, we know that `front` reads from the correct end in relation to `enqueue` and `dequeue`. Hence also the latter two act on the correct end.

Remark: In the end we know that `enqueue`, `dequeue` and `front` work correctly in relation to each other. But the implementation could still be reversed to how it was expected. As the queue is implemented with a list, this poses no problem and the datatype will still work as expected. However, when the queue would be implemented on top of a static datatype such as array, this could mean trouble. Adding a test 8, could check the absolute correct acting ends of all operations. This was not implemented in the current unit tests.

4.2 Dynamic memory handling in C

The given implementation of queues with 2-cell uses dynamic memory allocation in C. Memory handling functions are implemented. Therefore, when comparing the outcome of two independent queues, two sets of data have to be prepared. Operations such as `dequeue` will deallocate dynamic memory hence the data will not be available for the other queue operation. This was the case for Axiome 4 and 6.

4.3 Application of unit tests to provided datatype

The implemented unit tests where applied to the provided datatype `queue` (implemented with a 2-cell list). Here, the unit tests all passed.

4.4 Provoking unit test fails

Several modifications on the given datatype were tested to provoke unit test fails. Modifications were always tested on the whole chain of unit tests. Each modifcation stated below was tested separate.

1. Test 1, `isEmpty)` was modified to always return `FALSE`.

2. Test 2, `isEmpty` was modified to always return `TRUE`.

3. Test 3, in `enqueue` the function call to the list function was removed.

4. Test 4, in `dequeue`, the function call was modfied to show stack behaviour, entries were removed on the previous to last list position.

5. Test 5, in `front`, the function call was removed and instead the address to a hardcoded `int` value returned. This resulted in a compiler warning as a local stack variable was passed on. It worked however to provoke a fail for axiom 5.

6. Test 6, for both `enqueue` and `dequeue` the position of action was changed: `enqueue` added at first list position, while `dequeue` removed the previous to last.

4.5 Memory leaks

Valgrind was used to detect and mend memory leaks, both in ‘pass’ and ‘fail’ cases.

References

[1] L.E. Janlert and T. Wiberg. *Datatyper och algoritmer*. Studentlitteratur, 2000.