Test Report Team 11

```
{

    CU_initialize_registry();
    CU_pSuite unitTests =CU_add_suite("Enhetstester för refmem", NULL, NULL);

    CU_add_test(unitTests, "retain           unitTest",test_retain );
    CU_add_test(unitTests, "release          unitTest",test_release );
    CU_add_test(unitTests, "rc               unitTest",test_rc );



    CU_pSuite memTests = CU_add_suite("Enhetstester för funktioner där memleak ej ska finnas", NULL, NULL);

    CU_add_test(memTests, "allocate          unitTest",test_allocate );
    CU_add_test(memTests, "allocate_array    unitTest",test_allocate_array );
    CU_add_test(memTests, "deallocate        unitTest",test_deallocate );
    CU_add_test(memTests, "cleanup           unitTest",test_cleanup );
    CU_add_test(memTests, "shutdown          unitTest",test_shutdown );
    CU_add_test(memTests, "default destructorunitTest", test_default_destructor);

    CU_pSuite cascadeTests = CU_add_suite("Tests testing cascadelimit", NULL, NULL);
    CU_add_test(cascadeTests, "set_cascade_limit unitTest",test_set_cascade_limit );
    CU_add_test(cascadeTests, "get_cascade_limit unitTest",test_get_cascade_limit );
    CU_add_test(cascadeTests, "test cascadeLimit functionality", test_cascade);

    CU_pSuite edgeCases = CU_add_suite("Testing edge cases for gcov", NULL, NULL);
    CU_add_test(edgeCases, "empty Cleanup", test_empty_cleanup);
    CU_add_test(edgeCases, "rc with null", test_rc_with_null);



    CU_basic_run_tests();
    CU_cleanup_registry();
```

Figure 1: Unit testing main-file

```
int main()
{

  //Init testing
  CU_initialize_registry();

  CU_pSuite suite_base = CU_add_suite("Base functions", NULL, NULL);
  CU_add_test(suite_base, "add_merch", test_add_merch);
  CU_add_test(suite_base, "remove merch", test_remove_merch);

  CU_add_test(suite_base, "replenish merch", test_replenish_merch);

  CU_add_test(suite_base, "edit merch", test_edit_merch);
  CU_add_test(suite_base, "add to cart", test_add_to_cart);
  CU_add_test(suite_base, "calculate cost", test_calculate_cost);
  CU_add_test(suite_base, "checkout", test_checkout);

  CU_pSuite suite_static = CU_add_suite("Static functions", NULL, NULL);
  CU_add_test(suite_static, "warehouse_key_has_locations", test_key_has_locations);
  CU_add_test(suite_static, "is_shelf_available", test_shelf_available);
  CU_add_test(suite_static, "amount_to_add_available", test_amount_add_available);
  CU_add_test(suite_static, "is_merch_stocked", test_is_merch_stocked);
  CU_add_test(suite_static, "size_of_cart", test_size_of_cart);
  CU_add_test(suite_static, "is cart empty", test_is_cart_empty);
  // CU_add_test(suite_static, "remove cart_merch_with index", test_remove_cart_merch_with_index);

  //CU_add_test(suite_static, "get_cart_index", test_get_cart_index);
  CU_add_test(suite_static, "remove w/ cart", test_remove_with_cart);


  CU_basic_run_tests();
  CU_cleanup_registry();
```

Figure 2: Integration testing main-file

In our unit testing file test.c located in Team-11/proj/tes.c we tested each function in refmem.c individually separated from other functions. If we divide tests for individual functions and separate them from other functions the regression testing will be much more clearer. Incase one function stops to work we will be able to see that in the specific function test rather than spotting problems in all tests if we used that function in many places.

From figure 1 we can see that we had 3 suits from CUnit. One suit which tested functions from the given header file named unitTests. These tests could be tested using CU_ASSERTS to check if values and variables were correct at given places. Isolating these functions was tricky as we had to rewrite some functions to be able to perform CU_ASSERTS on given values. For example in the test_rc we needed to allocate an object using our allocate function, then we had to change the rf value for it to 1 rather than 0 without using our retain function:

```
}
void test_rc()
{
    first_info = NULL;
    last_info = NULL;

    cell1_t *c1 = allocate(sizeof(cell1_t), cell_destructor1);
    cell2_t *c2 = allocate(sizeof(cell2_t), cell_destructor2);
    objectInfo_t *c1_Info = ((obj *) c1 - sizeof(objectInfo_t));
    objectInfo_t *c2_Info = ((obj *) c2 - sizeof(objectInfo_t));
    c1_Info->rf = 5;
    c2_Info->rf = 3;
    CU_ASSERT_TRUE(rc(c1)== 5);
    CU_ASSERT_TRUE(rc(c2)== 3);
    c1_Info->rf = 10;
    c2_Info->rf = 5;
    CU_ASSERT_TRUE(rc(c1)== 10);
    CU_ASSERT_TRUE(rc(c2)== 5);
    c2_Info->rf = 0;
    c1_Info->rf = 0;
```

Figure 3: test_rc exampe

Even if these test isn't as readable as it would be as if we would use retain, the purpose of the test is much better. Incase we would perform regression tests after changing our program and retain would stop to work, there would only be errors in the test_retain() rather than test_rcl()+test_retain().

We use pointer arithmetic in all of our functions. To make sure there was no “luck” with pointer arithmetic that made each test pass, we made two cell structs with different sizes that we ran on all unit tests testing individual functions. This way we can prove our code works for more structs.

```
typedef struct cell1 cell1_t;
typedef struct cell2 cell2_t;


struct cell1
{
    cell1_t *cell;

    char *string;
    int k;
    int s;
    size_t mitocondria;
    int i;
    char *string2;
};

struct cell2
{
    cell2_t *cell;
    char *string;
    int k;
};
```

Figure 4: Cell structs in unit tests

We had one suite which tested functions that allocated and deallocated memory. This suite was named “memTests”. Functions that were tested here was allocate(), allocateArray(), deallocate(), cleanup(), shutdown(), default_destructor(). Incase the tests ran as they should and the we saw 0 leaks after using valgrind we say that these test passed.

We had one suit for special cases of functions that wasn't tested in the rest of the suits. The purpose of this suit is to achieve a 100% coverage in refmem.c. If we manage to achieve a gcov we prove that all our code has been tested in at least 1 case. This suit is called “edgeCases”. We run our unitTests using valgrind with **make testValgrind.**

In our integration tests we copied assignment two from one of our group members. We went into all files and changed everywhere it said calloc/malloc to instead use allocate() from refmem.h. This lead to that we also needed to add retain() after each allocate(), and to find where these allocates were freed and instead use release() from refmem.h. After we had changed all code to use our library instead, we tried to see if the already made tests warehouse_tests.c could be run using our library. We ended up running these tests without any memory leakage. We run our integration tests with valgrind using **make demo_run.**

most nasty bugs

1. Our shutdown function did not work with destructors such as cell destructors which released both a cell and the cell it was pointing to. This was because both cells would be inserted into our linked list and when the first cell was destroyed with the cell destructor and we tried to destroy the second cell its memory was already freed.
2. When we tried using the destructor stored in the object info on an object nothing happened. This had to do with our code never extracting the destructor and instead found a whole different place in the memory with something completely different.
3. When we reach our cascadelimit we do not store the pointer to the object that was next in queue to be deallocated. This resulted in us having to use the shutdown function to avoid memory leaks in the cases we reached the cascade limit.
4. In the beginning we did not reduce the reference counter by one until after we checked to see that it was not 0. This was solved by having the release function reduce the reference counter by 1 and then comparing it to 0 afterwards.
5. We also had some bugs regarding the default destructor as we used double pointers in a wrong way.