EE/CprE/SE 492 - sdmay21-10
Dynamic Programming Based FPGA Hosted Control-Flow Integrity (CFI) Report 4

March 1st - March 15th
Client: Akhilesh Tyagi
Faculty Advisors: Zelong Li, Ananda Biswas

Team Members

Gregory Wendt - Meeting Scribe
Cole Schumacher - Meeting Facilitator
Nickolas Mitchell - Chief Engineer (FPGA)
Sam Henley - Chief Engineer (Software)
Maxwell Wangler - Test Engineer
Tristan Duyvejonck - Report Manager

Past Weeks Accomplishments

Over the past two weeks, our team has continued developing an LLVM pass as well as working towards a complete FPGA solution. Sam resolved the issue where the system would run out of memory space when building LLVM from source. He worked closely with our faculty advisors to continue developing LLVM passes. Nick resolved the issue of figuring out to directly program an FPGA board. He was able to download a student version therefore eliminating the issues using the virtual lab computers presented. Nick also began translating our java race logic code into VHDL as well as implemented a few test programs from previous semesters. Lastly, Greg and Max continued their work translating our Java test suite over to C.

Pending Issues

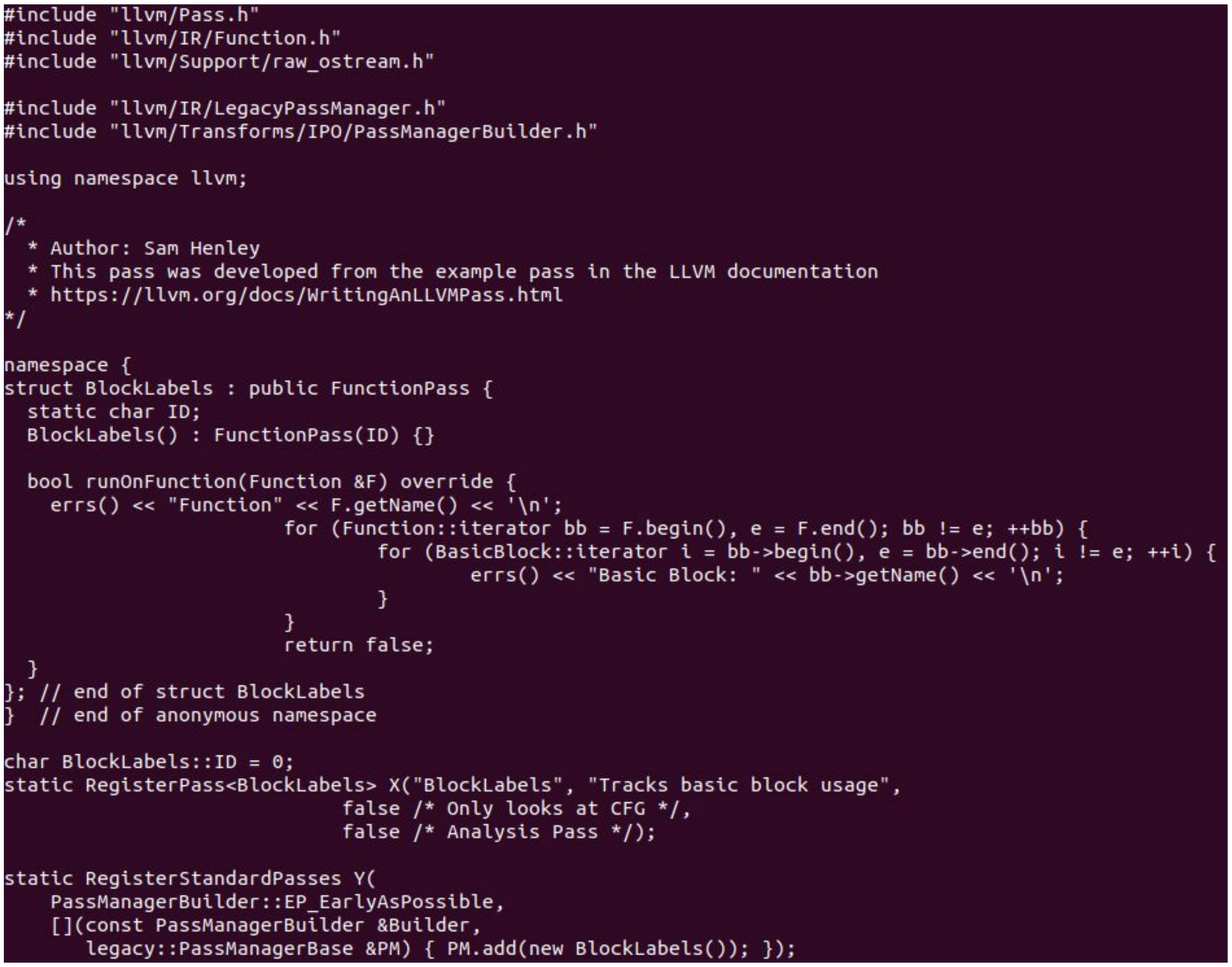
- Work towards a better understanding of LLVM compilers
  - Debug issues with LLVM
  - Figure out how to get the generated labels (compile-time and runtime)
- Finish VHDL solution
  - Test VHDL code
- Brainstorm ways to send data to the FPGA board.
  - FPGA solution will need two string inputs for the program to use
  - Could use meta data from a header file or research other methods.
- Test 8-bit string inputs

Individual contributions

NAME	Individual Contributions (Quick list of contributions. This should be short.)	Hours this week	HOURS cumulative
Gregory Wendt	Worked with Max translating Java code and test to C code and updated website	4	14
Cole Schumacher	Learned more about sharing software we are using, looked deeper into FPGA, coordinated team meetings.	2	12
Nickolas Mitchell	Resolved issue of directly programming FPGA board. Started translating Java code into VHDL. Worked with test programs to help in translation of code.	12	32
Sam Henley	Debug issues with building LLVM source, develop a pass	10	19
Maxwell Wangler	Worked with Greg in translating the Java code and test to C code. Creating a test framework for C	4	12.5
Tristan Duyvejonck	Unable to contribute	0	3

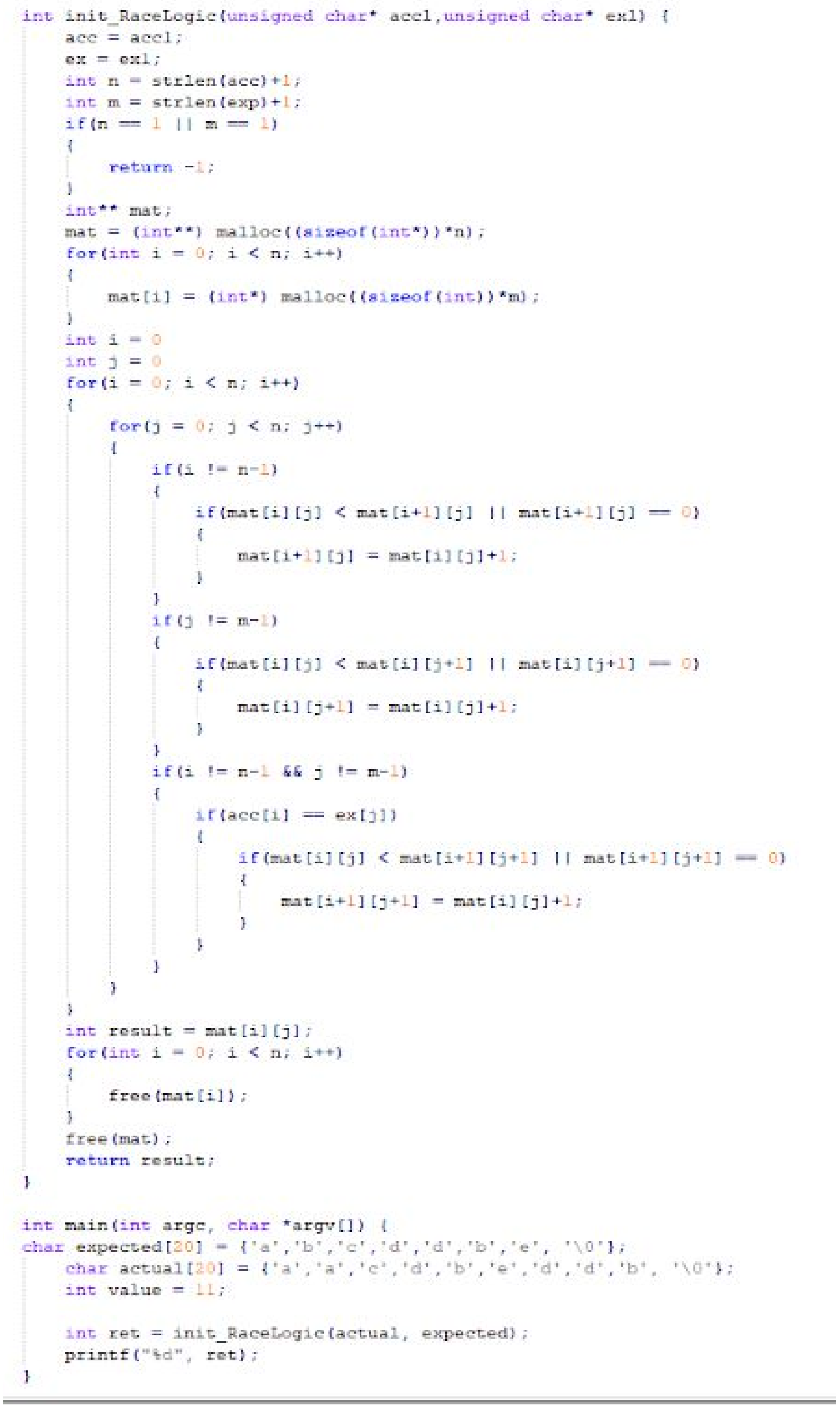
Plans for the Upcoming Report

- Gregory
    - Work with Max to test 8bit instructions
    - Confirm translation Java to C
- Cole
    - Help Nick with FSM and VHDL code
- Nickolas
    - Use FSM to continue work on FPGA solution
        - Test VHDL code
    - Look into ways to send and receive strings on FPGA board
- Sam
    - Debug issues with loading a pass
    - Test passes on sample programs
- Maxwell
    - Continue testing 8bit instructions
    - Confirm Java Translation
- Tristan
    - Work with Sam developing LLVM passes



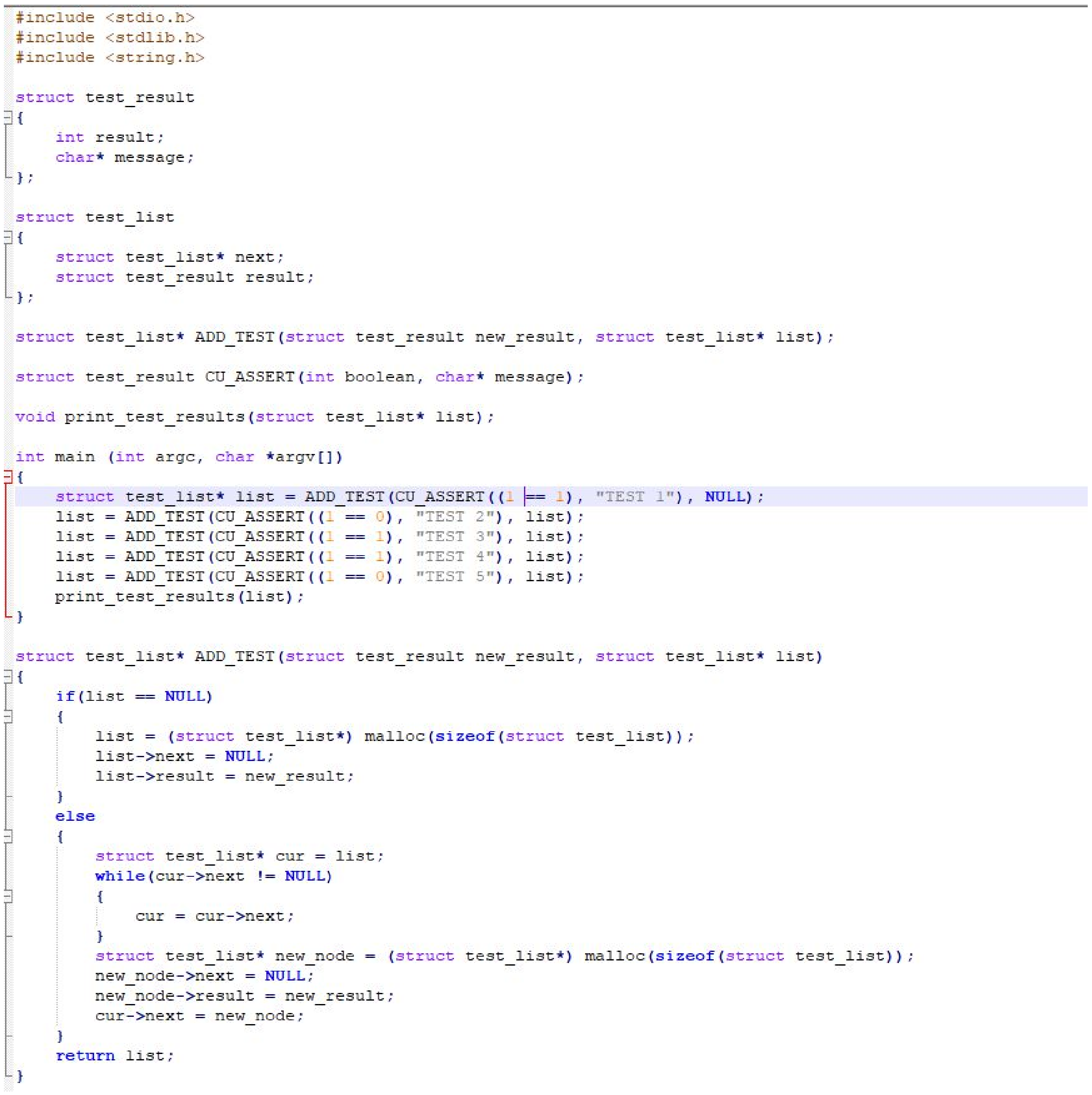
```cpp
#include "llvm/Pass.h"
#include "llvm/IR/Function.h"
#include "llvm/Support/raw_ostream.h"

#include "llvm/IR/LegacyPassManager.h"
#include "llvm/Transforms/IPO/PassManagerBuilder.h"

using namespace llvm;

/*
  * Author: Sam Henley
  * This pass was developed from the example pass in the LLVM documentation
  * https://llvm.org/docs/WritingAnLLVMPass.html
*/

namespace {
struct BlockLabels : public FunctionPass {
  static char ID;
  BlockLabels() : FunctionPass(ID) {}

  bool runOnFunction(Function &F) override {
    errs() << "Function" << F.getName() << '\n';
                        for (Function::iterator bb = F.begin(), e = F.end(); bb != e; ++bb) {
                                for (BasicBlock::iterator i = bb->begin(), e = bb->end(); i != e; ++i) {
                                        errs() << "Basic Block: " << bb->getName() << '\n';
                                }
                        }
                        return false;
  }
}; // end of struct BlockLabels
}  // end of anonymous namespace

char BlockLabels::ID = 0;
static RegisterPass<BlockLabels> X("BlockLabels", "Tracks basic block usage",
                             false /* Only looks at CFG */,
                             false /* Analysis Pass */);

static RegisterStandardPasses Y(
    PassManagerBuilder::EP_EarlyAsPossible,
    [](const PassManagerBuilder &Builder,
       legacy::PassManagerBase &PM) { PM.add(new BlockLabels()); });
```

Figure 1: LLVM BlockLabels Pass



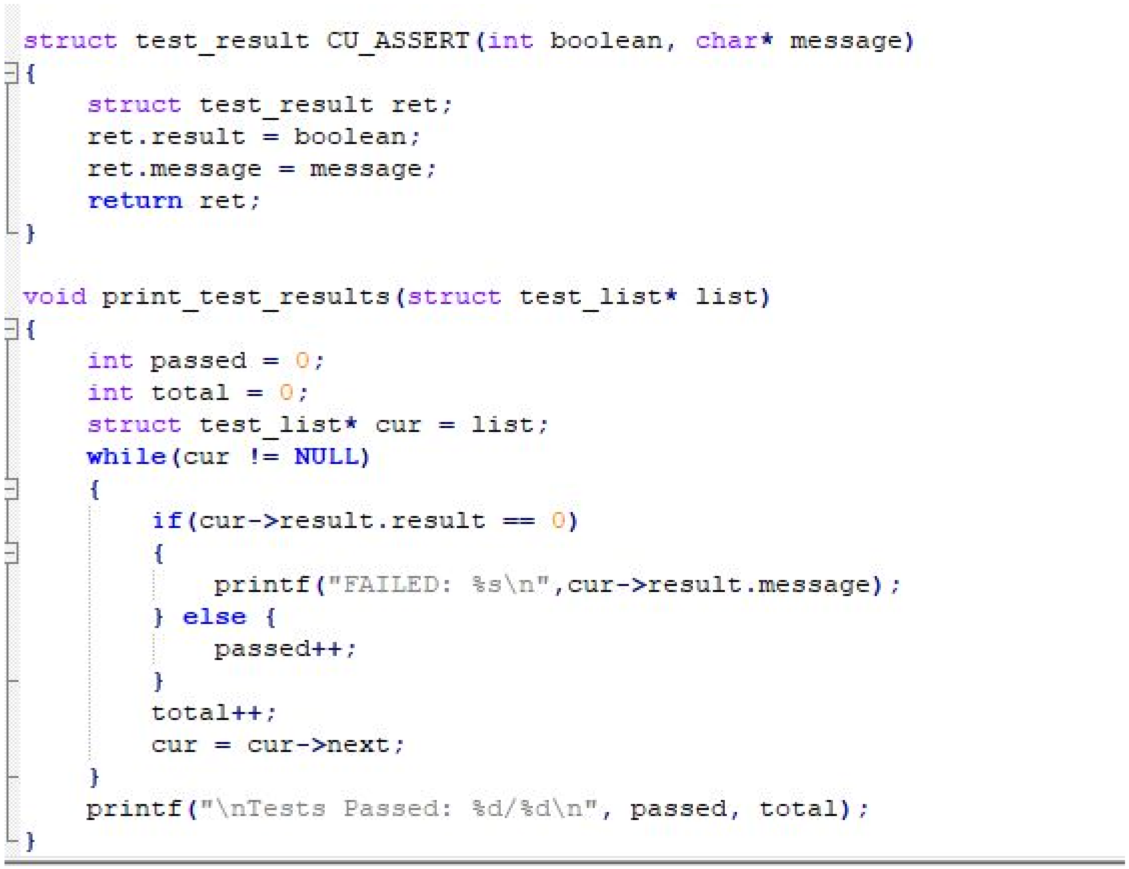
```c
int init_RaceLogic(unsigned char* acc1,unsigned char* ex1) {
    acc = acc1;
    ex = ex1;
    int n = strlen(acc)+1;
    int m = strlen(exp)+1;
    if(n == 1 || m == 1)
    {
        return -1;
    }
    int** mat;
    mat = (int**) malloc((sizeof(int*))*n);
    for(int i = 0; i < n; i++)
    {
        mat[i] = (int*) malloc((sizeof(int))*m);
    }
    int i = 0
    int j = 0
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            if(i != n-1)
            {
                if(mat[i][j] < mat[i+1][j] || mat[i+1][j] == 0)
                {
                    mat[i+1][j] = mat[i][j]+1;
                }
            }
            if(j != m-1)
            {
                if(mat[i][j] < mat[i][j+1] || mat[i][j+1] == 0)
                {
                    mat[i][j+1] = mat[i][j]+1;
                }
            }
            if(i != n-1 && j != m-1)
            {
                if(acc[i] == ex[j])
                {
                    if(mat[i][j] < mat[i+1][j+1] || mat[i+1][j+1] == 0)
                    {
                        mat[i+1][j+1] = mat[i][j]+1;
                    }
                }
            }
        }
    }
    int result = mat[i][j];
    for(int i = 0; i < n; i++)
    {
        free(mat[i]);
    }
    free(mat);
    return result;
}

int main(int argc, char *argv[]) {
char expected[20] = {'a','b','c','d','d','b','e', '\0'};
    char actual[20] = {'a','a','c','d','b','e','d','d','b', '\0'};
    int value = 11;

    int ret = init_RaceLogic(actual, expected);
    printf("%d", ret);
}
```

Figure 2: RaceLogic moved to C



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct test_result
{
    int result;
    char* message;
};

struct test_list
{
    struct test_list* next;
    struct test_result result;
};

struct test_list* ADD_TEST(struct test_result new_result, struct test_list* list);

struct test_result CU_ASSERT(int boolean, char* message);

void print_test_results(struct test_list* list);

int main (int argc, char *argv[])
{
    struct test_list* list = ADD_TEST(CU_ASSERT((1 == 1), "TEST 1"), NULL);
    list = ADD_TEST(CU_ASSERT((1 == 0), "TEST 2"), list);
    list = ADD_TEST(CU_ASSERT((1 == 1), "TEST 3"), list);
    list = ADD_TEST(CU_ASSERT((1 == 1), "TEST 4"), list);
    list = ADD_TEST(CU_ASSERT((1 == 0), "TEST 5"), list);
    print_test_results(list);
}

struct test_list* ADD_TEST(struct test_result new_result, struct test_list* list)
{
    if(list == NULL)
    {
        list = (struct test_list*) malloc(sizeof(struct test_list));
        list->next = NULL;
        list->result = new_result;
    }
    else
    {
        struct test_list* cur = list;
        while(cur->next != NULL)
        {
            cur = cur->next;
        }
        struct test_list* new_node = (struct test_list*) malloc(sizeof(struct test_list));
        new_node->next = NULL;
        new_node->result = new_result;
        cur->next = new_node;
    }
    return list;
}
```

Figure 3: Test framework part 1



```c
struct test_result CU_ASSERT(int boolean, char* message)
{
    struct test_result ret;
    ret.result = boolean;
    ret.message = message;
    return ret;
}

void print_test_results(struct test_list* list)
{
    int passed = 0;
    int total = 0;
    struct test_list* cur = list;
    while(cur != NULL)
    {
        if(cur->result.result == 0)
        {
            printf("FAILED: %s\n",cur->result.message);
        } else {
            passed++;
        }
        total++;
        cur = cur->next;
    }
    printf("\nTests Passed: %d/%d\n", passed, total);
}
```

Figure 4: Test framework part 2