C++ Automated Test Suite

Senior Design Final Documentation

Intolerable Optimists

Charles Parsons Christopher Smith Jarod Hogan

March 23, 2014

Contents

List of Figures

List of Tables

List of Algorithms

Mission

This system will automatically test C++ programs by traversing through directories and subdirectories to locate the cpp and tst files. We will also generate basic tests per user request.

Document Preparation and Updates

Current Version [1.1.0]

Prepared By:
Charles Parsons
Christopher Smith
Jarod Hogan

Revision History

Date	***Author***	***Version***	***Comments***
2/2/12	*Charles Parsons*	*1.0.0*	*Initial version*
3/4/12	*Christopher Smith*	*1.1.0*	*Edited version*
3/21/12	*Jarod Hogan*	*1.2.1*	*Automatic Test generator added*
3/22/12	*Christopher Smith*	*1.2.2*	*Next Phase*

1

Overview and concept of operations

This system is for professors to test their students' programs against test data contained in text files. The results of these tests will then be written to a text file for th user to evaluate the tested programs performance.

Title: Dr. Logar's User Story As a professor I want to have a program compile, run, test, and evaluate the tests run on a program submitted to me by my students so that I no longer have to run each test case individually and evaluate them by hand. The progam should compile and run a program against all test cases contained in files ending in ".tst" within the current directory or within a subdirectory. In addition, it will evaluate the passes and fails of the program and log this in a text document for user review. Titile: Revisions to make life easier. It is also desired that this test suite runs all test files contained in a test directory against each program submitted by a student. Each student will have their own directory where the result file should be stored. Title: Automatic tests. As I am quite I would like to have the test suite generate its own tests so that I don't have to. It will store these tests in the tests subdirectory and generate the answer file for each test.

1.1 Scope

The purpose of this project is to create a program that will run another program against test documents and output the results into another file forthe user. The program is specifically geared toward computer science professors for their use in grading student programs. This program will compile and run the submitted program imputting the test data. The results, which are saved to a text document, are evaluated and and percentage of pass/fail is also computed and saved in the document.

1.2 Purpose

This system is designed in order to make it easier for professors to grade their students programs. The test files need only be in the same directory (or within a subdirectory) as the program to be compiled and run. Also, the output is to be detailed enough so that the user knows exactly which test cases passed and which failed.

1.2.1 Program Compiler

Though rather simple to code this is a mojor component because it depends on which system the program is running as to how the program is compiled. This program is made to run using the gcc command.

1.2.2 Test File Location

Also a simple but necessary component to our system. The program is going to run every file ending in ".tst" within the same directory as the progam itself. Therefore, searching the current working directory as well as every subdirectory is very important to ensure every test case is run.

1.2.3 Test Case Evaluation

Here is the most critical part of the program. This is where the tests are read in by our program and the results evaluated. Once evaluated the results are stored in a text file for the user to review. We will test various different possbile directory orderings as well as the ability to run against varying types of test cases.

1.3 Systems Goals

The goals of our system is to successfully compile and run a program with the test files in the directory. Then to output the result of the tests in a text file. It is also our goal to generate our own test cases for the user and generate cooresponding answer files.

1.4 System Overview and Diagram

The system flow is pretty simple. The program will walk through directories searching for test cases to run student programs on. With this resent update we have a specific file directory we expect. This program will be placed in the root directory. In this root directory will be a subdirectory for each student as well as a tests subdirectory, which will naturally store the tests. There will also be a golden.cpp in the root directory. This program, if prompted, will generate test cases and store them in the tests subdirectory and will use the golden.cpp to generate anwer files which will be placed in the root directory. Afterwards, the program will walk through the students directories and run their program against all the tests and give them a grade.

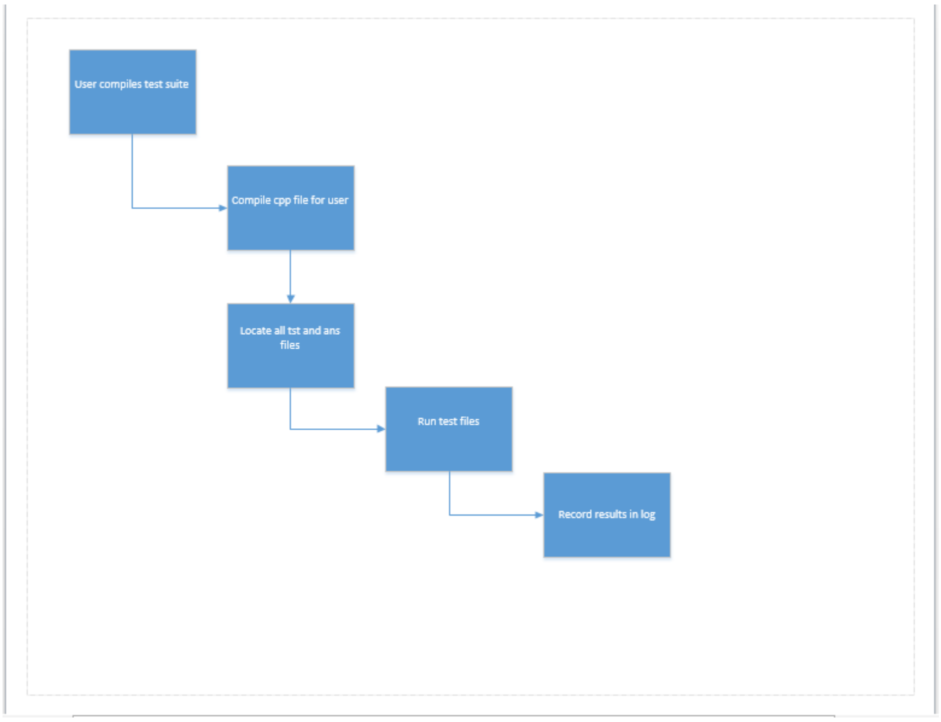

Figure 1.1: A sample figure .... System Diagram

1.5 Technologies Overview

There was no need for wide range of technologies in this specfic program. However, we have made extensive use of some software to enable us to be productive. Firstly we have used GitHub for version control. We used Trello to keep everybody on track and give an easy place to comunicate job assignments. Finally, but most importantly we used the Linux operating system on the Fujitsu tablets to do all our programing.

7C0	hexadecimal
3700	octal
11111000000	binary
1984	decimal

Table 1.1: A sample Table ... some numbers.

2

Project Overview

We are a team of three: Charles, Chris, and Jarod. Together we are designing a test suite. It will be able to run tests against students programs and grade them by comparing their answers to those generated by the testing suite, which will also be able to generate its own tests should the user decide.

2.1 Team Members and Roles

Charles Parsons is the Product Owner. He does all interaction with the customer, in this case Dr. Logar. He also makes sure the team delivers what he has agreed to with the customer.

Christopher Smith is our Scrum Master. He keeps the team on tract making sure we meet our deadlines. He is also responsible for making sure the Agile process is adhered to as much as possible.

Jarod Hogan is the Technical Lead. He is responsible for making sure the program is being done in the way agreed upon by the team. He also communicates any major problems or roadblocks in accomplishing the task as far as the software is concerned.

2.2 Project Management Approach

This project will be managed using the Scrum 'flavor' of the Agile process. We will follow the requirements of daily scrum meetings, as well as evaluation meetings throughout each sprint and especially at the end. Trello will be used to help manage each person and their tasks. GitHub is to be utilized to keep each team member up to date as to what each person has accomplished.

2.3 Phase Overview

The First phase of our testing suite was to have it running a C++ program against all test cases contained in the current directory as well as each subdirectory. The test cases ended with the extension .tst. It also generated at file containing the results of each test run and stored them in a file.

The Second phase of our testing suite was to have it generate its own tests and answers and then grade multiple programs based on their accuracy to match the answer files. This required a bit of reworking since we now need to compile and run many C++ programs and run them against tests contained in a specific directory.Also each program is stored in its own directory.

2.4 Terminology and Acronyms

Intentianally left blank

3

User Stories, Backlog and Requirements

3.1 Overview

This system is for professors to test their students' programs against test data contained in text files. The results of these tests will then be written to a text file for th user to evaluate the tested programs performance.

Title: Dr. Logar's User Story As a professor I want to have a program compile, run, test, and evaluate the tests run on a program submitted to me by my students so that I no longer have to run each test case individually and evaluate them by hand.

The progam should compile and run a program against all test cases contained in files ending in ".tst" within the current directory or within a subdirectory. In addition, it will evaluate the passes and fails of the program and log this in a text document for user review.

3.1.1 Scope

The purpose of this project is to create a program that will run another program against test documents and output the results into another file forthe user. The program is specifically geared toward computer science professors for their use in grading student programs. This program will compile and run the submitted program imputting the test data. The results, which are saved to a text document, are evaluated and and percentage of pass/fail is also computed and saved in the document.

What scope does this document cover? This document would contain stakeholder information, initial user stories, requirements, proof of concept results, and various research task results.

3.1.2 Purpose of the System

This system is designed in order to make it easier for professors to grade their students programs. The test files need only be in the same directory (or within a subdirectory) as the program to be compiled and run. Also, the output is to be detailed enough so that the user knows exactly which test cases passed and which failed.

3.2 Stakeholder Information

There are only two main stake holders in this project. The first is Dr, Logar, our customer, who naturally expects a working progam. The second stake holder is this team who greatly desires an A in this course.

3.2.1 Customer or End User (Product Owner)

Dr. Logar is our customer and end user in this case.

3.2.2 Management or Instructor (Scrum Master)

Cris Smith is our Scrum Master. He plans the time line we are to follow as well as the scrum meetings

3.2.3 Investors

There are no Investors.

3.2.4 Developers –Testers

There is not set developer(s) or tester(s). We divide up the work as evenly as possible. After running our own tests on our code we are sure to test the finished product to ensure integrating the code was successful.

3.3 Business Need

This program is aimed at reducing the amount of time a professor must personally spend on each program by automating the test phase.

3.4 Requirements and Design Constraints

The main constraint of this program is that it must run on a Linux based operating system and be used to compile C++ programs. Other constraints are that this program does require a basic knowledge of command line compiling.

3.4.1 System Requirements

The system requirements are that only C++ programs be used with this project on a Linuz machine.

3.4.2 Network Requirements

There are no network requirements as everything can be done offline.

3.4.3 Development Environment Requirements

Our developement enviornment is Linux using simple text editors with g++ compile command.

3.4.4 Project Management Methodology

The stakeholders might restrict how the project implementation will be managed. There may be constraints on when design meetings will take place. There might be restrictions on how often progress reports need to be provided and to whom.

- What system will be used to keep track of the backlogs and sprint status?
- Will all parties have access to the Sprint and Product Backlogs?
- How many Sprints will encompass this particular project?
- How long are the Sprint Cycles?
- Are there restrictions on source control?

3.5 User Stories

3.5.1 User Story #1

Title: Dr. Logar's User Story As a professor I want to have a program compile, run, test, and evaluate the tests run on a program submitted to me by my students so that I no longer have to run each test case individually and evaluate them by hand.

3.5.1.a User Story #1 Breakdown

This user story is very straight forward yet complex. The first major requirement is we compile and run a program automatically. Secondly the output must be evaluated and a report prepared for the user.

3.5.2 User Story #2

Titile: Revisions to make life easier. It is also desired that this test suite runs all test files contained in a test directory against each program submitted by a student. Each student will have their own directory where the result file should be stored.

3.5.2.a User Story #2 Breakdown

The important information contained here is how we traverse the file structure and where we find test cases and store the results.

3.5.3 User Story #3

Title: Automatic tests. As I am quite busy I would like to have the test suite generate its own tests so that I don't have to. It will store these tests in the tests subdirectory and generate the answer file for each test.

3.5.3.a User Story #3 Breakdown

This is pretty simple and straightforward. All that needs doing running a program against randomly generated test cases.

3.6 Research or Proof of Concept Results

Intentially left blank

3.7 Supporting Material

Intentinally left blank

4

Design and Implementation

Below is discussed the major components of the test suite. There are only two major components in the testing suite so far. The first is the Directory traversing and the second is test generating and evaluation.

Algorithm 1 Overall Algorithm

```
Compile .cpp file
Locate .tst and .ans files
while Locate .tst and .ans files do
   Run .tst file
   if .tst File passes then
      $K \Leftarrow K + 1$
   end if
   $D \Leftarrow D + 1$
end while
$PercentagePassed$ ⇐K / D
```

4.1 Major Component #1

As mentioned above the first major component is the directory traverse algorithm. This is the heart of the testing suite as well as where all the real power lies. It is here where we talk a strole through the directory and each subdirectory to search for every students .cpp program to compile and run it.

4.1.1 Technologies Used

The majortechnologies used were the Fujitsu tablets running on the Linux OS. We also used gedit and vim as our text editors.

4.1.2 Component Overview

The system traversies the subdirectories seeking a C++ program to run and compile as well as test files for evaluation.

4.1.3 Phase Overview

This phase searches for *.cpp files to compile and run.

4.1.4 Architecture Diagram

Compile a *.cpp file passed as a command line argument. Run that executable against all *.tst files within the current directory and all of its sub-directories.

4.1.5 Data Flow Diagram

Input is redirected from the *.tst file and output is redirected to a *.out file. The *.out file is compared to the corresponding *.ans file. The results are printed in a *.log file.

4.1.6 Design Details

This is where the details are presented and may contain subsections. Here is an example code listing:

```
#include <stdio.h>
#define N 10
/* Block
 * comment */

void findTestFile(string file, const string &program, int &pass, int &fail, ofstream &fout
{
    ifstream fin;
    struct dirent **namelist; //structure in dirent.h stores the file name
                              //and the file id number
    int n;
    string ans_file;
    string log_line;
    string tmp;
    string actual_out;
    int i;
    //scans teh current directory for all
    //types stores how many are found in n and the names in namelist
    n = scandir(file.c_str(), &namelist, 0, alphasort);
    if(n == -1)
        return;
    //starts at the second position since
    //the first to files found are the . and .. directories
    for(i=2; i<n; i++)
    {
        //checks if the file found is a .tst
        if(check_if_file( namelist[i] -> d_name ))
        {

            tmp = string(namelist[i]->d_name);
            ans_file = tmp.substr(0,tmp.find_last_of('.') ) + ".out";

            if(test(program, file,ans_file,tmp,log_line))
            {
                log_file(fout, log_line,"pass");
                pass++;
            }
            else
            {
                log_file(fout, log_line, "fail");
                fail++;
            }

        }
        //if the file is not a .tst but is a .ans
        //or . .. it skips those files
        else if(check_if_dot(namelist[i] -> d_name ))
        {
            continue;
        }
        //goes into this else if the file is a directory
        else
        {
            tmp = string(namelist[i] -> d_name);
            findTestFile(file + "/" + tmp, program, pass, fail, fout);
        }
    }
    for(i = 0; i < n; i++)
        delete []namelist[i];
    delete []namelist;
}
```

This code listing is not floating or automatically numbered. If you want autonumbering, but it in the algorithm environment (not algorithmic however) shown above.

4.2 Major Component #2

The second component of the system is the test generation and evaluation.

4.2.1 Technologies Used

The majortechnologies used were the Fujitsu tablets running on the Linux OS. We also used gedit and vim as our text editors.

4.2.2 Component Overview

The system traversies the subdirectories seeking a C++ program to run and compile as well as test files for evaluation.

4.2.3 Phase Overview

This phase searches for *.cpp files to compile and run.

4.2.4 Architecture Diagram

It is important to build and maintain an architecture diagram. However, it may be that a component is best described visually with a data flow diagram.

4.2.5 Architecture Diagram

Compile a *.cpp file passed as a command line argument. Run that executable against all *.tst files within the current directory and all of its sub-directories.

4.2.6 Data Flow Diagram

Input is redirected from the *.tst file and output is redirected to a *.out file. The *.out file is compared to the corresponding *.ans file. The results are printed in a *.log file.

5

System and Unit Testing

This section describes the approach taken with regard to system and unit testing.

5.1 Overview

Originally, the test suite was run against the test files verifying it ran the tests against the .cpp file correctly. The primary tests to make sure our walk through the directory was done correctly. This means we exaust all directories and run all .tst files.

5.2 Dependencies

The program requires *.tst and *.ans files that are either in the same directory as the *.cpp file or in a sub-directory of that directory.

5.3 Test Setup and Execution

Dr. Logar provided several test files and directories for the program to process. The executable was copied into each of the directories containing a *.cpp file to test against. We verified all these cases were handled successfully. In addition to the tests provided by Dr. Logar we also a few tests checking some of the directory bountries. Such tests include running without any subdirectories, running with all subdirectories empty, having nested subdirectories each with varing numbers of test cases ranging from zero to ten.

6 Development Environment

The basic purpose for this section is to give a developer all of the necessary information to setup their development environment to run, test, and/or develop.

6.1 Development IDE and Tools

Our developmentenviornment was the Linux package installed on our tablets curtiosy of the ACM. We also used vim and gedit as text editors for this project.

6.2 Source Control

We started out using bitbucket for our source control but switched over to gitbucket for more uniformity with the class.

6.3 Dependencies

There were no major dependencies for this project. The main dependency is the OS we operate on. The code for the test suite is Linux specific.

6.4 Build Environment

The build enviornment uses the gcc command to complie the program. A make file has been made so the user need only type make.

6.5 Development Machine Setup

Intentionally left blank. This is because we are dependent on the Linux distribution from the ACM.

7

Release – Setup – Deployment

The release date will be the final turn in date for this program at the end of the semester. We have multiple smaller releases where we send out what has been completed at the end of each sprint.

7.1 Deployment Information and Dependencies

Main dependencies are the Linux operating system, C++ programs, and a basic knowledge of command line compiling. The code will be compiled using a makefile so all the user need do is to pull the code off of github onto their linux device.

7.2 Setup Information

The test suite should reside in the root directory in which the student and test directories are contained. There also needs to be a golden.cpp in the root directory to generate the answer files to each test file generated by the suite.

7.3 System Versioning Information

There will be at least three major versions tothis program. The first is a program which compiles and runs a given .cpp file and runs it against all test files in a directory and subdirectory. The second version consists of some automatic test generating as well as searching for .cpp files in subdirectories belonging to each student of a class.

8

User Documentation

Here is contained a brief set of instructions for how to install and use our test suite.

8.1 User Guide

The use of this suite is quite simple. Simply place our code into the root directory of the files you wish to run against your program. Keep the programs within their respective subdirectories and all the test files within the tests subdirectory. Then use the makefile to run the program.

8.2 Installation Guide

Installation is very straightforward. After obtaining perission from us you need only to pull the code off of our github group used for version control.

8.3 Programmer Manual

Intentionally left blank. Unsure what to place here.

9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

10

Class Documentation

Acknowledgement

Thanks

Bibliography

[1] R. Arkin. *Governing Lethal Behavior in Autonomous Robots.* Taylor & Francis, 2009.

[2] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press, Cambridge, MA, June 2005.

[3] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[4] V. Lumelsky and A. Stepanov. Path planning strategies for point mobile automation moving amidst unknown obstacles of arbirary shape. *Algorithmica*, pages 403–430, 1987.

[5] S.A. NOLFI and D.A. FLOREANO. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines.* A Bradford book. A BRADFORD BOOK/THE MIT PRESS, 2000.

[6] Wikipedia. Asimo — Wikipedia, the free encyclopedia. `http://upload.wikimedia.org/wikipedia/commons/thumb/0/05/HONDA_ASIMO.jpg/450px-HONDA_ASIMO.jpg`, 2013. [Online; accessed June 23, 2013].