ECE297 Milestone 0
Understand your Design Tools

"Give me six hours to chop down a tree and I will spend the first four sharpening the axe."
— Abraham Lincoln

Assigned on Wednesday Jan. 8
Due on Tuesday, January 21 @ 7pm

In-lab demo = 0.75/1
Autotester = 0.25/1
Total marks = 1/100

1 Objectives

The purpose of this milestone is to introduce you to the design tools (NetBeans or Eclipse, gdb, Valgrind) that you will be using throughout this course. Detailed tutorials are provided on the course website to help you get started. Make sure you understand what each tool does and how to use it (ask your TA if you don't) because you will use all of them in this course and you will suffer if you don't have a good grasp on things from the start.

1	Use **Netbeans or Eclipse** to develop and debug a C++ project

- Open a project & edit code
- Compile your project in debug, debug_check and release modes

2	Manage changes with **Git**

- Commit changes
- View history & compare revisions
- Change to a particular revision
- Share commits with your colleagues

3	Use the **debugger**, **debug_check** configuration and **Valgrind** to find errors

- Run the debugger and examine your program's state
- Select and run the debug_check build configuration to find memory errors
- Run valgrind on an executable to check for memory errors (invalid accesses or leaks)
- Interpret the output from these tools to find the location and cause of error

4	Submit your code to be automatically graded by the autotester.

- Run exercise to test code functionality
- Run submit to submit code for grading

You may have to demonstrate any of these tasks to a TA

2 Introduction

In this milestone, we give you a C++ program that approximates the centre of mass of a cuboid with an arbitrary density function. First, you'll modify the program slightly, while learning how to use Git. Then you'll find and fix the program's several (deliberately included) errors — valgrind and the debug_check build will be helpful here. Meanwhile, you will learn about Git and use it to record your progress of fixing the errors. Finally, you will demonstrate your knowledge of the tools to a TA.

3 Walkthrough

3.1 Environment Setup

All labs for ECE297 use the **EECG UG** Linux workstations — *not the ECF ones.* These machines are located in GB243, GB251, SF2102 and SF2204.

To setup the environment for ECE297 (e.g. to get access to the `ece297exercise` and `ece297submit` commands) you need to source the file `/cad2/ece297s/public/SOURCEME` This can be done automatically whenever you login by adding the command `source /cad2/ece297s/public/SOURCEME` to your `.cshrc` configuration file. To do so, first type open a terminal and run `gedit ~/.cshrc` This will open a graphical text editor. Find the line "`# Here's how to add a directory to your shell's path.`" and add the command `source /cad2/ece297s/public/SOURCEME` on a new line below it. That part of the file now should look like the following:

```
1  # Here's how to add a directory to your shell's path.
2  source /cad2/ece297s/public/SOURCEME
```

You must log-out and then log back in for these changes to take effect!

Now, try run the command `ece297exercise -h` It should print a message about how to use it, and *not* say 'command not found'.

~15mins

To setup remote access to the UG workstations see the "ECE297 Quickstart Guide: VNC".

3.2 Set up the project

To setup the project, run `ece297init 0` The command prints several details about what it did, but for now, lets focus on the folder it created, `~/ece297/work/milestone0`, and the contents within.

This document is written assuming you are using the Netbeans Integrated Development Environment (IDE). However, if you prefer you can instead use the Eclipse IDE which has even more features than NetBeans – the exact structure of its menus is different, but the basic functionality is the same. Open Netbeans by typing `netbeans` in a terminal window and pressing enter. Open the project with Netbeans by clicking through `File` → `Open Project` and navigating to the `milestone0` directory/project.

~30mins

If you don't remember how to use/set-up Netbeans, refer to "ECE297 Quickstart Guide: Netbeans" now.

~30mins

If you would prefer to instead learn the Eclipse IDE, refer to "ECE297 Quickstart Guide: Eclipse" now.

3.3 Modifying a Simple Program & Using Version Control

In this task, you will modify some code, and learn the basics of of version control.

Look at the current main.cpp file; this is where your program starts execution. Currently the polynomial $f(x, y, z) = 7x^3 + \frac{1}{2}y^2 + z^5$ is defined and it describes the mass-density distribution over a solid 3D body. After that, the polynomial is passed to the function `compute_center_of_mass` along with the x, y and z dimensions of a cuboid ((x,y,z) = (5,6,10) in the program currently). You are asked to do the following actions

1. Modify the polynomial $f(x, y, z)$ such that the coefficient of x is 5 instead of 7.
2. Define another polynomial $g(x, y, z) = x + 2y^2 + 3z^3$.
3. Compute the center of mass of a cube with (x,y,z) = (5,5,5) and mass-density distribution $g(x)$.
4. Print out the newly computed center of mass (do not remove existing print statement and make sure the new print statement is exactly the same as the old one).
5. Delete the last print statement which outputs "Program Finished!" – use delete or backspace on your keyboard, don't just comment it out.

Modify your C++ project with the modifications itemized above.

Now, open the ECE297 Quickstart Guide for Git which will teach you about version control, and walk you through the rest of the steps for this task.

Once you have finished the Git Quickstart Guide, you should be familiar with the basics of Git, and be ready to move onto the next task of this milestone. Here are some example questions to test your understanding. You don't have to write down the answers anywhere, but a TA will ask you similar questions in your lab sessions, and they expect you to demonstrate a good understanding of these fundamental Git commands. If you're not sure what the answer to one of these questions is, try to find the answer in the Guide, ask a friend, consult the Internet, or ask a TA. You will be allowed to answer/demonstrate using the command-line or Netbeans.

How do I ...

- see what changes I have made?
- see the commit history? Where are the commit IDs?
- make a commit?
- share my changes with my colleagues?
- get my colleagues' changes?
- fix a commit when `pull` fails?
- see a file as it was committed for a particular commit?
- undo all my uncommited changes?
- revert a file to how it was in particular commit?

Be sure to know how to answer the questions above

In the following task, you will debug memory problems with your code and commit a new revision every time you fix one of the bugs.

3.4 Find and debug memory problems with appropriate tools

Even though the project that you are given compiles and runs successfully, its coding is quite sloppy. There are a number of memory problems with this piece of software. Below we describe tools and techniques that can help you find these memory problems.

3.4.1 Build configurations

Your project includes three different build configurations – each configuration sets different options that control how your program is turned into an executable program. In NetBeans you select which configuration is active with the drop down shown in Fig. 1. You should always do a clean and build after changing configurations, to make sure the program you are running or debugging matches the selected configuration.

The three build configurations available in this milestone are:

- `debug`: This configuration is compiled to be easier to understand when run in the debugger; each line of code is directly translated to machine instructions.

- `debug_check`: This configuration adds extra checks that your program will execute as it runs. These checks will immediately stop your program and produce a report of the problem in the output tab if they find an error condition, such as trying to dereference a NULL pointer or access an element of an array that doesn't exist. These checks slow your program down, so it will run slower than the `debug` configuration.
- `release`: The compiler produces an *optimized* executable that runs faster than the `debug` configuration. If you run this code in a debugger it will be harder to understand, as the machine code may perform operations in a different order than your source code, and some function or variables may have been optimized away. (e.g. some short functions may have been *inlined* so that the instructions they should perform are instead inserted into the calling function).

3.4.2 Using the debug_check build configuration

Select the `debug_check` configuration, perform a `Clean and Build` and run the program. Examine the output generated to see if there is a memory error, and if so, where. In Fig. 1 you can see that running this configuration has identified that line 12 of math_utils.cpp is accessing entry [4] of an array that only has elements [0] to [2] (this bug is shown as an example only; it is not in your version of the code).

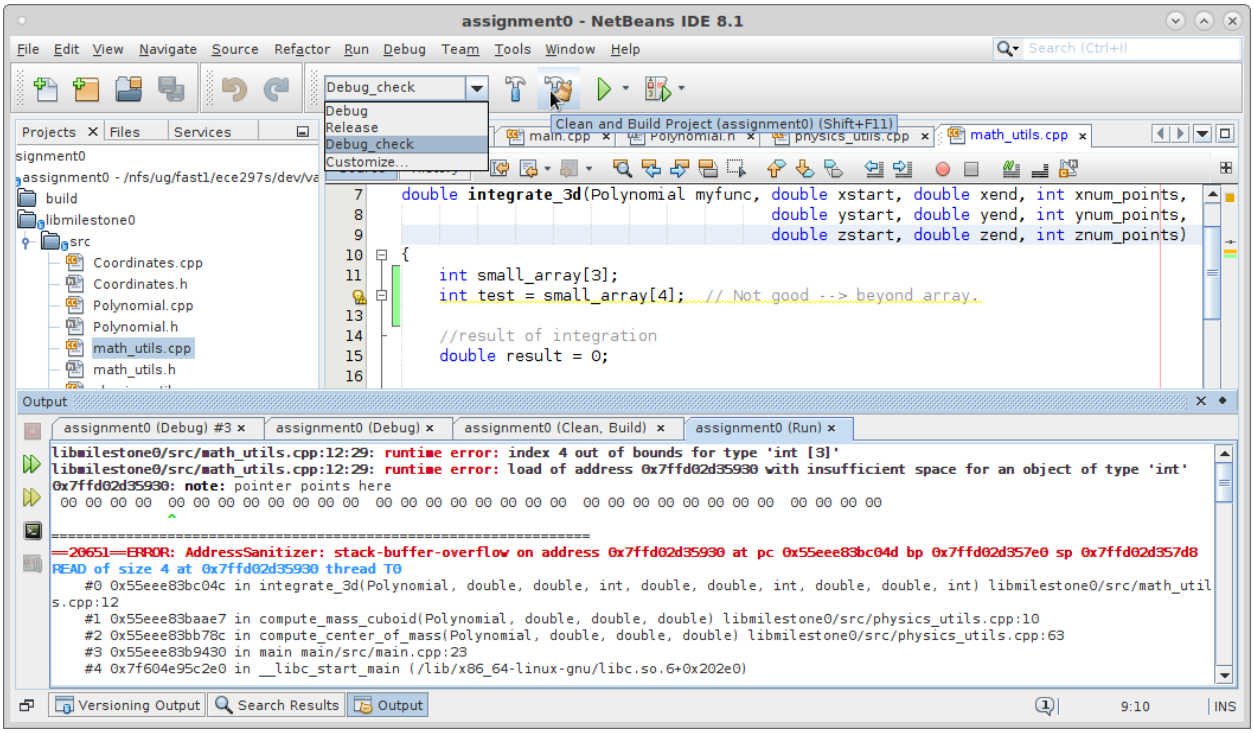

Figure 1: Using the debug_check build configuration.

3.4.3 Using Valgrind

We can also use Valgrind to debug these memory leaks and invalid memory accesses. Valgrind can run on any executable program and it will check not only your code for invalid memory accesses,

but also any library functions you call. These checks slow your program down, generally by even more than the using *debug_check* build configuration does.


~40mins

To learn how to use Valgrind and what it does, go read "ECE297 Quickstart Guide: Valgrind". You can also try out the examples there.

You can run Valgrind either from within NetBeans (by making a new configuration) or from the command line with a terminal window. Running from a terminal window is a bit quicker to set up in this case. Valgrind works best with a debug build so select the `Debug` configuration and do a clean and build. Open a terminal window and `cd` to the directory containing the executable version of the center of mass program. If you aren't sure where that is, hold your cursor over the project name in the NetBeans Projects window and it will show you the directory holding this project – this is your *working copy* directory. The executable program will be in this (root of the working copy) directory, and it is called `milestone0`. run `valgrind --leak-check=full milestone0` to have valgrind check your milestone0 program as it is running.

3.4.4 Using the debugger

You can stop your program's execution at any point and examine the state of variables using the debugger. This is a key technique in efficiently finding and fixing bugs.


~40mins

If you are not familiar with how to start and use the debugger, go through the ECE 297 Debugger Quick Start Guide.

3.5 Fixing the bugs and commiting the changes

By using these 3 tools you can find the memory bugs (there are approximately 4 of them, but we won't tell you exactly how many). Keep debugging until valgrind and debug_check report no errors. After **every bug** you fix, you must `commit` and `push` your code with a clear and concise commit message. In your lab session, your TA will ask you to jump between revisions that you have in your repository and you will have only the commit messages to guide you.

Solve all the memory bugs, while `commiting` and `pushing` your changes to the repository *after each bug is fixed.*

3.6 Using the autotester

It is important to always test any code you create or modify. Even though you may think your code is correct this is rarely the case. Code (especially new code) is rarely bug free. It is well worth the time and effort required to *prove* (to yourself) that the code is correct by testing it.

To help make this process easier, you are provided with an autotester. The autotester will exercise your code on a set of public testcases and inform you if the result is correct.

The public testcases are not complete.
The autotester runs some of the tests that we will use to mark your code's correctness, but additional non-public testcases will also be used in marking. Hence you should not only run the autotester but should also make additional tests to cover important corner cases.

The autotester is run from the command line. Example usage for this milestone is shown in Listing 1.

```
> ece297exercise 0 # Note: 0 is the milestone number
```

Listing 1: Autotester usage

The autotester requires the milestone number, and no further arguments are required. Optional arguments and additional information can be found by running `ece297exercise -h`

An example usage is shown in Listing 2

```
#In the milestone directory
# Executable and static library have already been built
> ls
Makefile     build           libmilestone0.a  nbproject
milestone0  libmilestone0  main

#Run the autotester
> ece297exercise 0
#Output trimmed ...
Test Summary: FAIL ( 3 of  6 failed)
  ExecTest FAIL ( 1 of  2 failed)
  Valgrind FAIL ( 2 of  4 failed)
```

Listing 2: Autotester example

3.7 Submitting your code

Do not wait until the last minute to submit.
Submit early so you have time to fix any problems. You can submit multiple times and only the last submission will be marked.

The final step of an milestone is to submit your code for marking. This is done with the `ece297submit`command from the command line. The usage of this command is shown in Listing 3.

```
> ece297submit 0
```

Listing 3: Submit usage

The submission script requires one argument, and can take several optional arguments. A list of and usage information for the arguments can be printed by running `ece297submit -h` We will mention one important optional argument here: the `--revision` argument. This allows you to specify a specific revision to submit, as it otherwise defaults to the HEAD of the local repository. This is useful in an emergency if you would like to submit a specific revision known to work.

An example usage is shown in Listing 4.

```
# Currently in the root of an working copy
> ls
Makefile     build           libmilestone0.a  nbproject
milestone0  libmilestone0  main

> ece297submit 0
# Various output showing configuration and process
# Be sure to check that the revision submitted is what you expect
```

Listing 4: Submission example

By default the submission script will perform the following actions automatically:

1. Get the current revision identifier of the local repository (HEAD)
2. Retrieve that version of your code from the your remote repository
3. Build your project in 'release' mode.

   This will make your program run faster than in 'debug' mode, which will be important for some CPU time tests later in the course.
4. Call the autotester (`ece297exercise`) on the resulting executable and static library.
5. Submit your project as a `tar` archive.

You can submit a milestone as many times as you want, until the submission deadline. Each submission will overwrite any previous submission.