Umeå University
Department of Applied Physics and Electronics

Linux as Development Environment 7.5 ECTS
5EL142 HT-16

Assignment 12 - Graphical Development Environments - Eclipse

Submitted 2017-08-13
Author: Lorenz Gerber, 20161202-2033 (`lorenzottogerber@gmail.com`)
Instructor: Sven Rönnbäck, John Berge, Björne A Lindberg

1 Introduction

The aim of this assignment was to learn about graphical development environments such as Eclipse. To achieve certain exposure to Eclipse, two practical exercises had to be conducted. First, the GTK+ electrotest application from lab 11 had to be setup for development in Eclipse and extended with a button to run the given test library linumtest.h. Eventual bugs had to be found and corrected. For the second part, a C++/GTK+ 'Mandelbrot' application had to be set up in Eclipse and extended with zoom in/out functionality.

The general idea about 'Graphical Ingegrated Development Environments' is to provide language tailored, advanced tooling for developing larger software projects. Some exampels of such tools are code completion, graphical debugging, graphical library managment, dependency checking, versioning etc. Usually, such graphical IDE's such as eclipse simply intgerate available console tools with a graphical possibility to choose the options.

2 Method

Here it was chosen to install 'eclipse neon' using the 'Oomph' installer. The 'Ooomph' installer is a separate piece of software that provisions the installation of 'eclipse' development environments. After startup of 'oomph', 'Eclipse CDT' the eclipse version for C/C++ development is chosen, then the whole IDE is installed automatically.

2.1 electrotestgtk

Setup of the electrotestgtk project for development was conducted according to the instructions given on the course homepage in the following order:

1. 'Setting up projects in Eclipse'
2. 'Configuration of Eclipse Projects for GTK+-2.0'
3. 'Installation and Use of own Libraries in Eclipse'

The first step, setting up a project was followed as described. For the GTK+-2.0 configuration, some additional steps were needed that the librarires were found correctly. Namely, '/usr/include/gtk-2.0' and '/usr/include/glib.2.0' had to be added in 'Properties' -> 'C/C++ Build' -> 'Settings' -> 'Cross GCC compiler' -> 'includes'. Similar, '/usr/share/glib-2.0' was added in 'Properties' -> 'C/C++ Build' -> 'Settings' -> 'Cross GCC Linker'-> 'Libraries'.

Installation of own libraries worked fine according the instructions. Prior to installation, new library versions with debugger flag (-g) set were generated.

After including the linumtest library, it was found that the component library did not adhere to the given function names, hence, this was corrected. Then the button and a callback function was added to the the electrotestgtk application.

Debugging was conducted by graphically setting breakpoints, starting the debug process by clicking the respective button and then using 'step over' and/or 'step into' functionality while inspecting the values of variables.

2.2 Mandelbrot

Setting up of the Mandelbrot project was correspondingly to the 'electrotestgtk' project except for choosing a 'C++' template. All the 'include' and 'Libarary' settings were done for the 'g++' build toolchain.

Initially the structure of the program was analyzed by debugger runs to step through the program. The ‘time_handler’ function was used to find and test suitable functions for implementing the zoom functionality in regard to mouse pointer positioning. Briefly, the pointer position was obtained as absolut position on the visible screen (‘gdk_display_get_pointer’). Then the ‘Gdk_window’ which in this has the same size as the main application window without the window decorations, was obtained using ‘gtk_widget_get_window’. Finally, the size of the Gdk_window was determined by ‘gdk_window_get_size’. Knowing the fixed size of the ‘GdkImage’ drawing area, the absolut position of the mouse pointer when within the GdkImage could be calculated.

For the zoom functionality it was established that a GdkImage length and height of each 800px corresponds the value ‘2’ in the fractal calculation algorithm. The input parameter for the fractal algorithm are the x/y value of the mid-image, which then corresponds to 400px/400px in GdkImage screen measures.

Zoom functionality was implemented as a single left click for zoom-in and single right-click for zoom-out. During testing, it was found that zoom-out should not change the center of the image but rather just zoom-out.

Four different compiler flags (O0, O1, O2, O3, Os) where tested for execution speed of the compiled code aswell as for binary size. Settings within Eclipse IDE where used to set the compiler flags. As a metric for code execution, the ‘time.h’ library was imported and ‘clock()’ was used to measure execution time for the first fractal calculated on application startup.

3 Results

3.1 electrotestgtk

As mentioned earlier, prior to debugging, the component library had to be recompiled with the correct function names. Then the linumtest program was run. It run without crash, however the test results indicated an error in the component library. Now the debugger was used to step trough the code and analyze the it’s function. For lab 6, I was in charge of another library, so I was not fully aware of how the component library works. The test failed in the ‘bTestComponent’ function. On stepping through, it was found that the second call to ‘e_resistance’ with the input parameter ‘100’ expects a return value of ‘2’, however e_resistance returns 1, hence the test fails. According to my understanding of how E12 resistances are calculated, 100 is part of the series, hence the result should indeed be 1. To amend the problem, the test function was modified to expect ‘1’ as return value. Now all tests passed and the GTK button turns green.

3.2 mandelbrot

The zoom functionality allows zooming in and out of the fractal by simply clicking the area of interest for zoom-in. Zoom-out keeps the current location. The zoom function makes use of the ‘zoom’ input argument of the ‘bCalc_Fractal’ function.

4 Discussion

Generally, setting up the grarphical IDE ‘Eclipse’ was straight forward and without any problems. I used ‘Eclipse’ already for Lab 10, altough with a external makefile setup.

Table 1 Contains the benchmark values for calculating the first fractal using different compiler optimization settings. Further, size of binary is also indicated.

flag	mean (sec)	binary size (K)
O0	3.21	502
O1	1.01	518
O2	0.84	521
O3	0.84	522
Os	1.38	514

Hence in this lab, the only new thing was to set up the libraries and includes for linking and compilation. It took a while in the beginning to understand which setting reflects which change on the auto-generated makefile. When understood, a graphical IDE allows for very quick and simple setup. Eclipse has since some time back also the feature (Oomph) that all settings can be written into configuration files so that for example a company could deploy a complicated development environment for a new employee within minutes. For me the striking advantage of an IDE such as Eclipse over a normal text editor or even a more advnaced setup with Emacs is the 'mouse over' feature that generate pop-ups, for example over macros, or functions with additional information. Also the code completion features are really neat.

4.1 electrotestgtk

Setting up the application from lab 11 in Eclipse was without problems. Using Eclipse for developing GTK+ applications seems to be a good choice as it simplifies for getting quick information about all the macros. Also graphical debugging feels like a serious productivity boos to me. I find this interesting as most of the tools and functionality available in Eclipse are basically the same as on the commandline.

4.2 mandelbrot

The run-time benchmarks showed that compilator optimization could speed up the tested calcuation three fold. From optimization level O2 to O3, no significant improvement was measured. The size of binary files showed that speed optimized ones where slightly larger in sequence of optimization level. The space optimized compilation was smaller than the all speed optimized, however still larger than the non optimized O0 version.

5 Conclusion

I used Eclipse a lot at work with Java, however only a few times so far with small one file C exercises. I enjoyed learning how to set up libraries in Eclipse and I was amazed how much code completion and mouse-over info boxes can help in C development - features that I so far only took for granted in Java development environments.

I did only see the document 'Hints on implementing Mandelbrot Fractal' after I already finished my version. Of course that is my fault, but if you want to improve something in the documentation, I think you could mention in the 'Övning 12 Grafiska Utvecklingsmiljöer' document that there are additional documents related to the 'mandelbrot' implementation. Then again, the lab was on a fair level even without the hints. I got some version running quite quick, but it took a while until I realized that I had to apply a scaling factor. Without it, the zooming got with every step of zoom more and more sensitive on choosing the zoom spot. As long as one stayed close to the image center it worked. Finally, after having found the but, zooming in and out the Mandelbrot kept me playing happily for at least half an hour (smile). That's what I call a rewarding exercise, really nice.