1 Software development tools

There exist a myriad of software development tools which help simplify the various tasks in creating robust and maintainable software. We present three tools in detail today:

- GIT version control system
  `git-scm.com/download/mac`
- Eclipse CDT IDE
  `www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/lunasr1`
- CUTE unit test framework
  `www.cute-test.com/`

Task 1 GIT tutorial

A version control system helps multiple developers working on the same code base. It keeps copies of previous code versions, which is helpful for debugging and software change analysis. It also allows to merge concurrent edits of text files. We use the version control system git for our tutorial.

Download and install git from `http://git-scm.com/download/mac`. Switch to the directory *ex-03/git-repo* before continuing. Git is a distributed version control system. This makes creating new repositories very easy. Transform this directory to a repository using the following command:

```
> git init --bare
```

The parameter *bare* advises git to create a repository without a working copy. This means that we cannot edit files from this folder. This is usually done on the server side, where the date is intended to be stored centrally.

In order to work in a working copy, we need to clone the repository into a separate working copy. We can do this by switching to the directory *git-clone1* and then issue the following command:

```
> git clone file://<full_path_to_bare_repo> .
```

In order for files to be tracked by the version control system, they need to be explicitly added to the repository. Use the following command to add the file *file1.c*.

```
> git add file1.c
```

The file is now part of the git repository. Changes made to this file need to be committed in order to be visible to other users. We can verify that the repository has pending changes using the following command:

```
> git status
```

Before we can commit this initial change, however, we are required to set our user id. This is necessary so that git can keep track of which user introduced which changes.

```
> git config --global user.name <preferredusername>
> git config --global user.email some@email.com
```

We now commit the initial state of the file. In order to do so, we need to provide an obligatory commit message, summarising our changes for other users to see.

```
> git commit file1.c –m "We just created this"
```

As mentioned before, git keeps track of all changes applied to all files. We can verify this using the log command. This will list our previous change, as well as all following changes.

```
> git log
```

We now propagate this data back to the central repository. The benefit of git is that we have full version control locally without having to connect back to the central repository. Only when we want to share all of our edited versions with the rest of the team we issue the push command:

```
> git push origin master
```

In a real world environment, git repositories are shared between multiple users. Each of them has a separate clone of the repository on their machine. In our test example, we create another clone of our own demo repository by switching to the directory *git-clone2* and executing the following command:

```
> git clone file://<full_path_to_bare_repo> .
```

We can now see that the file we previously added is already part of the clone. Edit the file and replace "John" by "Jane". Now commit the file using the following command:

```
> git commit file1.c –m "Changed name"
> git push origin master
```

If we switch back to the *git-clone2* directory, we see that the file has not been updated yet. We need to explicitly pull the new version from the central repo:

```
> git pull
```

This covers the basic functionalities of the git version control system. A more extensive demo can be found online:

`https://try.github.io/`

Task 2 Eclipse CDT tutorial

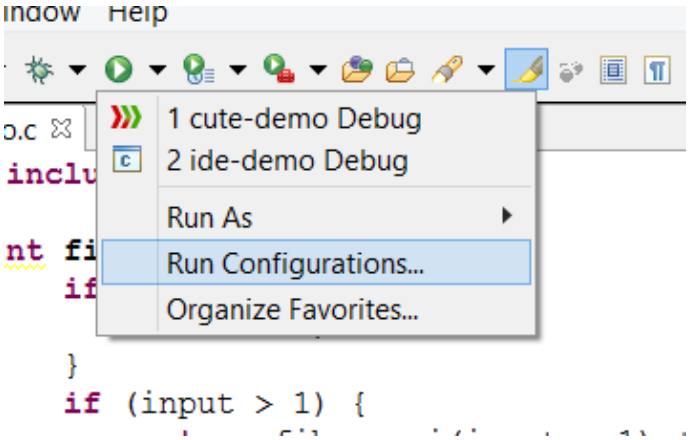
Eclipse CDT is an integrated development environment (IDE) for C++. IDEs are a play a major part in modern software engineering. Eclipse CDT contains a vast amount of features, which is why we can only take a cursory look at it in this tutorial. Launch Eclipse and select *ex-03/workspace* as your workspace. You find two projects in the workspace. Open the file *demo.c* in the project *ide-demo*.

In the Eclipse file editor, code warnings and errors are displayed as markers on the left. The function *fibonacci(...)* has a marker indicating that a return value may be missing. If you uncomment the return statement on line 10, the message disappears.

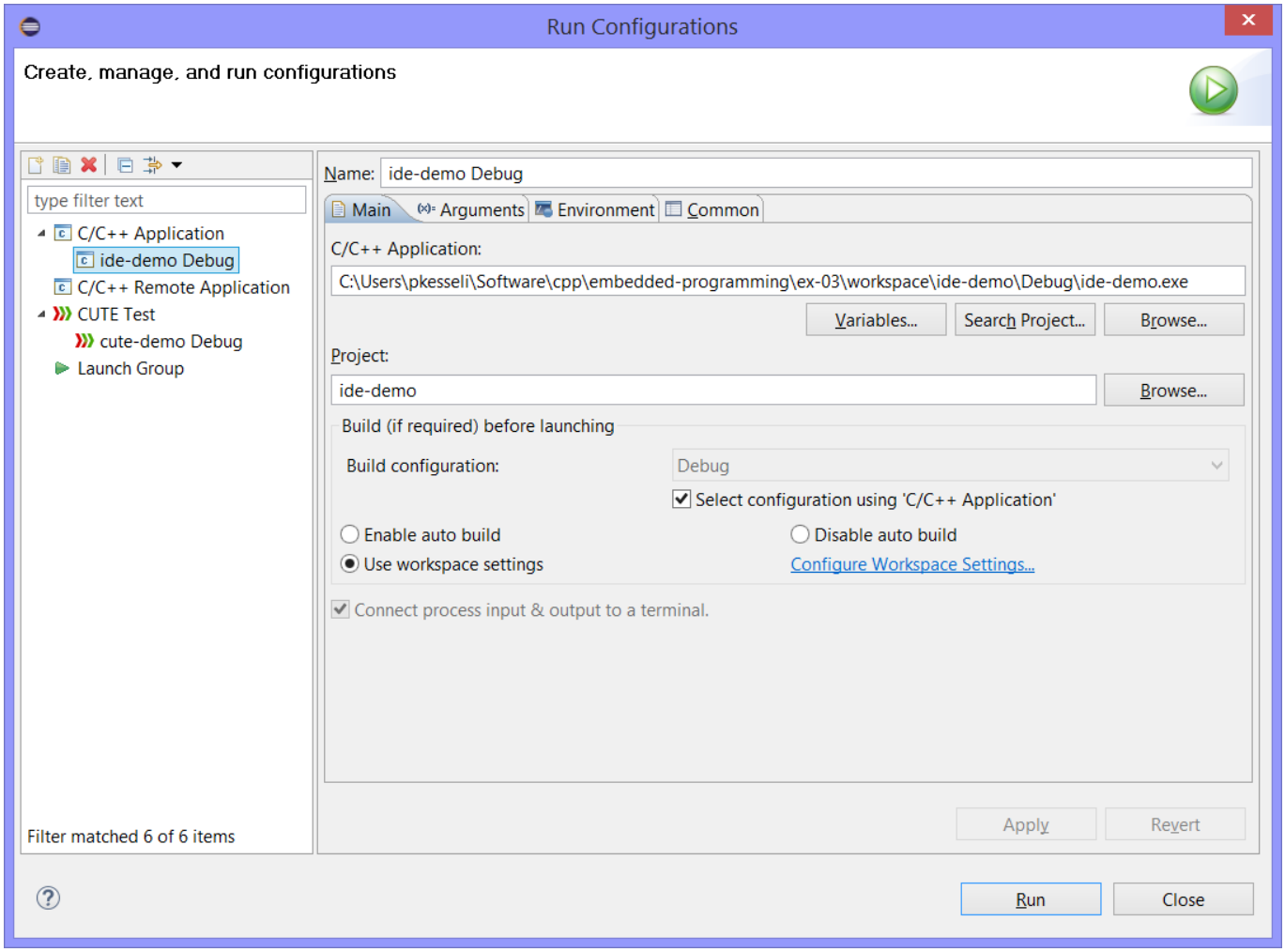
Another important feature is auto-complete. In Eclipse the key combination *Ctrl-Space* opens the auto-complete menu in any context. We would like to add the following statement to line 16:

```
printf("%d ", fibonacci(i));
```

Use the auto-complete feature for both *printf(...)* and *fibonacci(...)*. The key combination *Ctrl-B* then compiles the whole project. If the project builds successfully, continue to create a run configuration:



Configure the run configuration as shown in the picture and press “Run”:



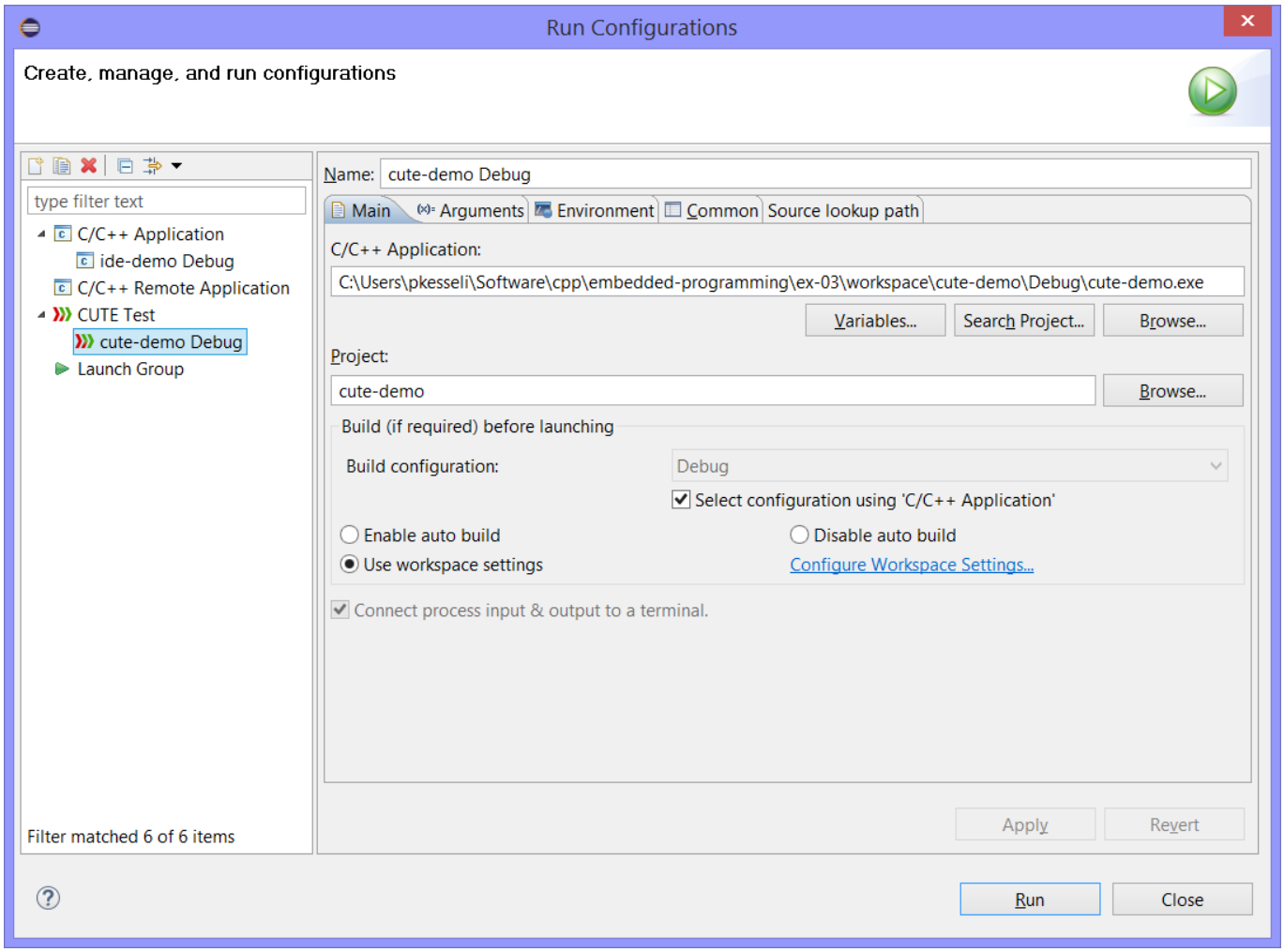
You should now see the following output in the “Console” window:

1 1 2 3 5 8 13 21 34 55

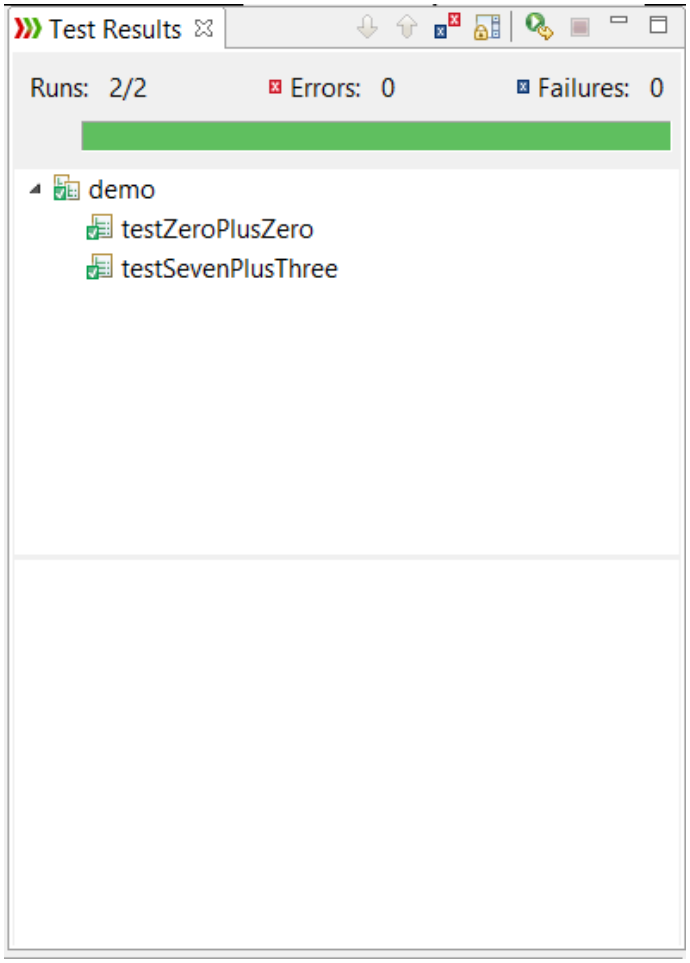
Task 3 CUTE tutorial

C++ Unit Testing Easier (CUTE) is a unit testing framework for C++. Unit tests are correctness specifications for written code. They are frequently used in modern software development processes. CUTE can be installed from the Eclipse Marketplace.

Click on "Help" and "Eclipse Marketplace…". In the appearing window, type "CUTE" and install the displayed plug-in. After a restart you are ready to run unit tests. In order to run the prepared unit tests, create the run configuration below:



Running this configuration will show the unit tests view, with all the executed test cases:



These tests are located in the file *demo.cpp*. Try to create three meaningful test cases for the function *fibonacci*. Register these test cases with the CUTE framework and re-run the unit tests.

2 Lookup tables

Lookup tables are an optimisation mechanism for functions with high computational complexity. They store a predefined amount of a functions input and output values in a data array. Instead of executing the computationally complex function at runtime, they then use the lookup table to retrieve previously calculated values.

This replaces a possibly very time-consuming algorithm executiong by a simple array index access. Drawbacks of this approach are that a significant amount of memory needs to be reserved for this lookup table. Since most embedded systems have limited resources, the size of the lookup table needs to be restricted as well. This leads to imprecision in the lookup table results.

Take a look at the file *lookup-table.c*. It contains a stub method to fill the lookup table array and one stub method to actually perform the lookup. The algorithm to be mapped is the *model_function(...)* function. All we need to do is map the real range $[0, 2 \cdot \pi]$ to integer array indexes $[0, 1000]$.

The code already generates a gnuplot data file which can be examined using the provided .plt gnuplot script. GNUplot can be downloaded from here: `http://gnuplot.info/`.

Task 1 Implement *fill_model_data(...)*

This function initialises our lookup table with the precalculated model values.

Task 2 Implement *fast_model_function(...)*

This function retrieves a precalculated model value from our lookup table. Once both functions are implemented, examine the result using gnuplot. What do you notice if you zoom in?

Task 3 Use <time.h> to measure the performance improvement.

Task 4 Implement *scaled_fast_model_function(...)*

As you can see in gnuplot, our lookup table approach lacks precision. One way to increase precision are range-based resolutions. In our original data table, we use the same resolution for the whole domain of the function. However, the function shows very low gradients in the ranges $[0, \frac{\pi}{6}]$, $[\frac{2\cdot\pi}{3}, \frac{7\cdot\pi}{6}]$ and $[\frac{5\cdot\pi}{3}, 2\cdot\pi]$. On the other hand, the ranges $[\frac{\pi}{6}, \frac{2\cdot\pi}{3}]$ and $[\frac{7\cdot\pi}{6}, \frac{5\cdot\pi}{3}]$ show high gradients.

We can improve precision by increasing the resolution of our data table in areas with high gradients and decreasing it in areas with lower gradients. This allows us to use the same amount of data points more efficiently. However, this also means that the entries in our data table are no longer indexed linearly. Instead, we need to store both input and output values in our array and search for the correct index in the array.

A trivial search algorithm for the array would be to check every array entry and select the one which is closest to our value. Since we can make sure that the entries in our datatable are at least ordered, a better approach is to implement a binary search algorithm.

Task 5 Which C type would be suitable to store our new data table?

Task 6 Look up the binary search algorithm online and describe it in pseudo code

Task 7 Implement the scaled lookup table model functions in our project

Task 8 Extra: Interpolation

An additional way to increase precision in a lookup table is linear interpolation. Instead of just selecting one array index, we select two and calculate the value to be returned as follows: $y_1 + (x_2 - x_1) \cdot (y_2 - y_1)$. Add interpolation to both of our lookup table model functions.