1 Multi-threading

This exercise delves into Multi-threading, which allows the program to perform sub-tasks concurrently. In order to create, manage, and terminate threads, we shall be using the POSIX API.

In the last lab session, we have implemented a linked list (if you haven't, an implementation is provided). We will be creating a program that iterates over several lists, calculating their total values via threads. Once the totals of each list is calculated, and the threads exit, we progress by adding up their results and print the aggregate value. Open the file, and perform the following tasks:

Task 1 Task 1: Implement the total function

Insert code in the total function.

Task 2 Task 2: Achieve Multi-threading

Modify `main`, calling the appropriate functions, such that the total of each list is calculated by a thread. Once all threads have terminated, calculate the aggregate. This requires understanding the code presented, so search online for documentation on the API functions.

2 Principles of Object Oriented (C++)

Polymorphism and inheritance are features provided by object oriented programming languages. Polymorphism enables an interface to interact with different types, whilst inheritance allows code extension, whereby properties of a parent/super class are also properties of the child/sub class.

Task 1 Task 1: Adding a Sub Class

Open `oo.cpp` and create a new subclass of an animal of your choice. Modify the main function such that you trigger the class's functionality via the `Animal` type interface.

Task 2 Task 2: Adding a new property

All animals should have a new field that denotes how many legs they have got. Modify the `Animal` class accordingly.

Task 3 Task 3: Extending the Dog class

Extend the Dog class with two further sub-classes, namely `Pet_Dog` and `Wild_Dog`. The `Pet_Dog` class should have a field `owner`, whilst the `Wild_Dog` should have an integer that indicates how big it's pack is.

3 Eclipse CDT tutorial

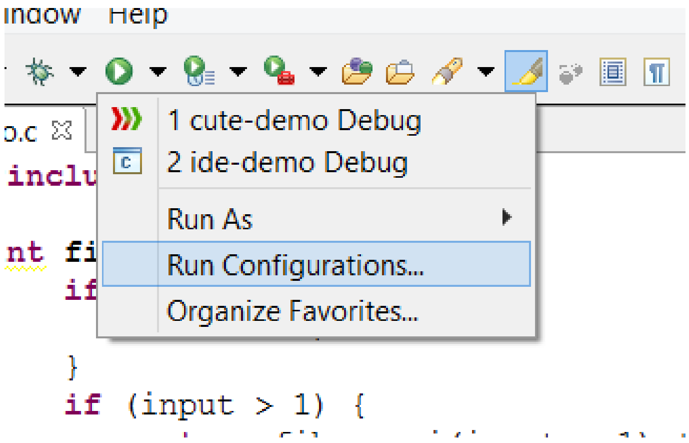
Eclipse CDT is an integrated development environment (IDE) for C++. IDEs are a play a major part in modern software engineering. Eclipse CDT contains a vast amount of features, which is why we can only take a cursory look at it in this tutorial. Launch Eclipse and select *ex-03/workspace* as your workspace.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

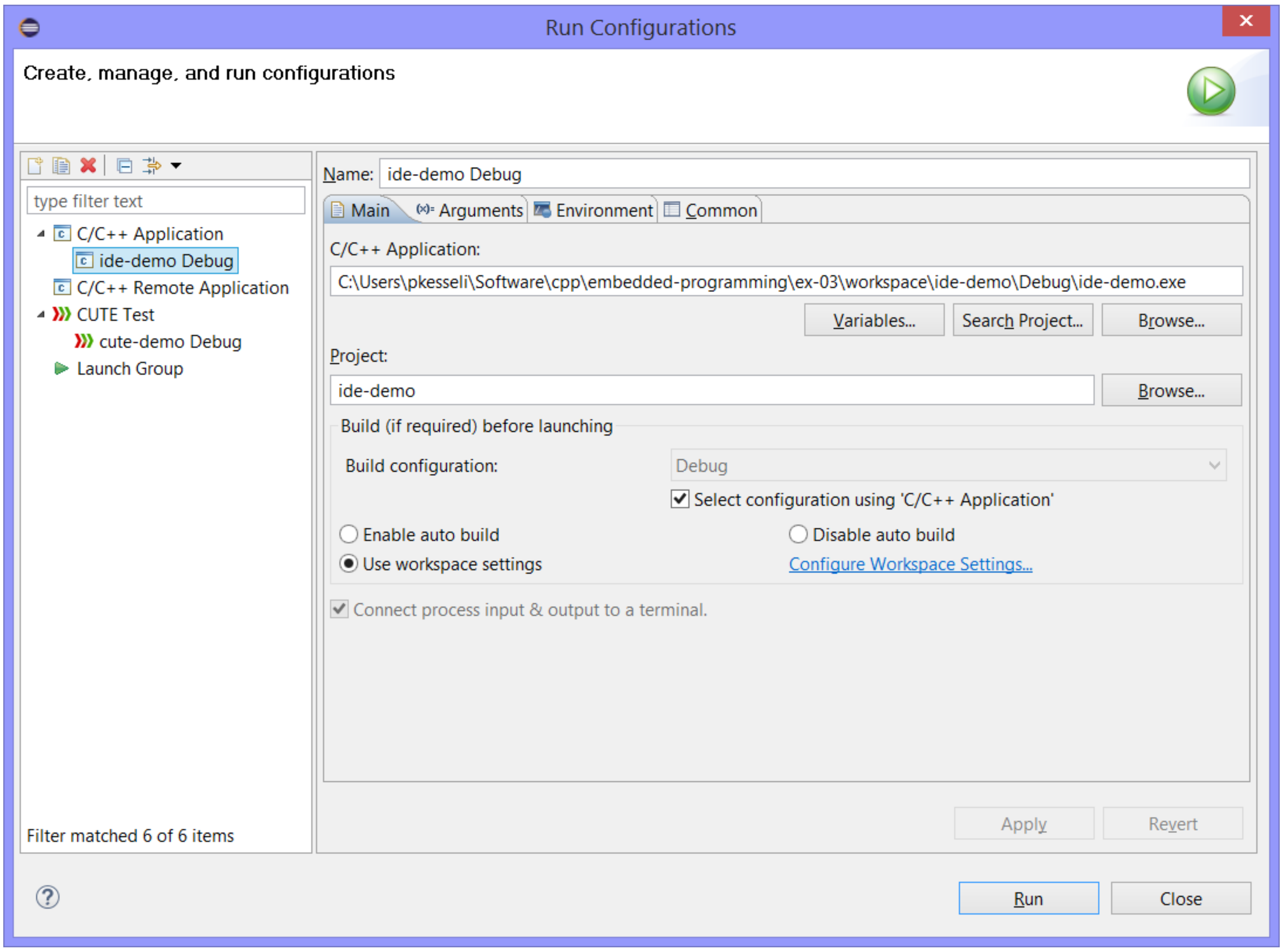
```
printf("%d ", fibonacci(i));
```

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:



```
incl
nt fi
  if
  }
  if (input > 1) {
```

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



Name: ide-demo Debug

C/C++ Application: C:\Users\pkesseli\Software\cpp\embedded-programming\ex-03\workspace\ide-demo\Debug\ide-demo.exe

Project: ide-demo

Build (if required) before launching

Build configuration: Debug

[✓] Select configuration using 'C/C++ Application'

( ) Enable auto build ( ) Disable auto build

(●) Use workspace settings Configure Workspace Settings...

[✓] Connect process input & output to a terminal.

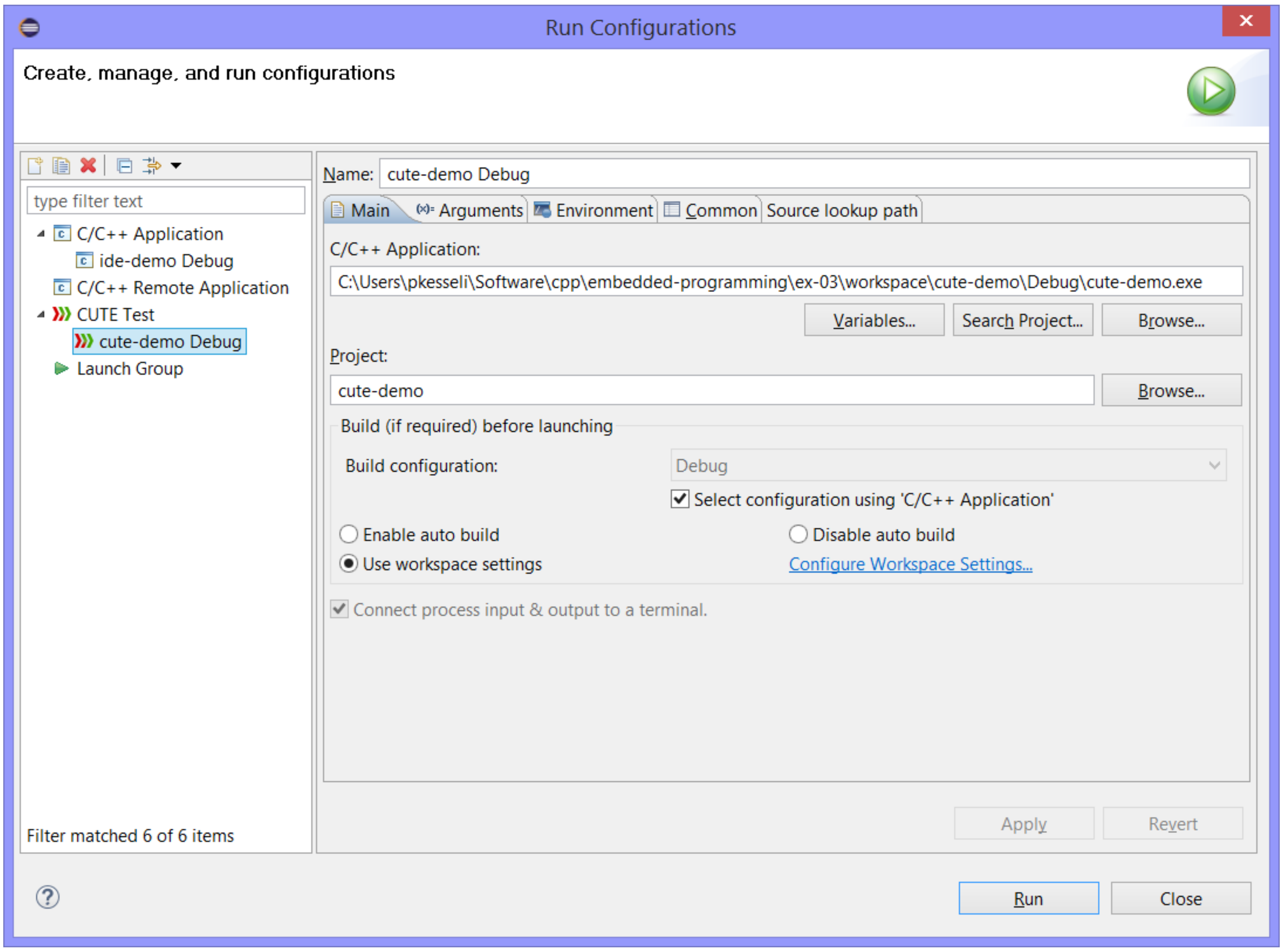
You should now see the following output in the “Console” window:

```
1 1 2 3 5 8 13 21 34 55
```

4 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:



Run Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
  - ide-demo Debug
- C/C++ Remote Application
- CUTE Test
  - cute-demo Debug
- Launch Group

Filter matched 6 of 6 items

Name: cute-demo Debug

Main | Arguments | Environment | Common | Source lookup path

C/C++ Application:
C:\Users\pkesseli\Software\cpp\embedded-programming\ex-03\workspace\cute-demo\Debug\cute-demo.exe

Variables... Search Project... Browse...

Project:
cute-demo

Browse...

Build (if required) before launching

Build configuration: Debug

[✓] Select configuration using 'C/C++ Application'

( ) Enable auto build ( ) Disable auto build

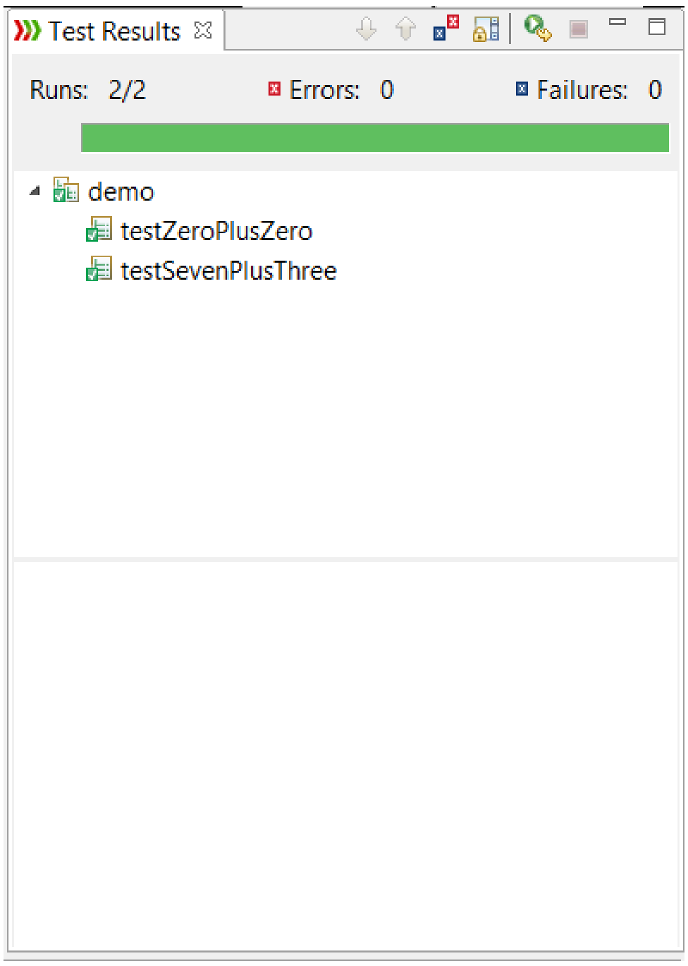
(●) Use workspace settings Configure Workspace Settings...

[✓] Connect process input & output to a terminal.

Apply Revert

Run Close

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.