3815ICT-Software Engineering Workshop 5

Activity 1

We will introduce you to systems that control the production of executables from several sources by presenting make (https://en.wikipedia.org/wiki/Make_(software)). There are several other alternatives, some of them proprietary. There exists also cmake (cmake.org/cmake-tutorial/) which aims to be multi-platform (en.wikipedia.org/wiki/CMake). Some of this are very commonly used in robotics as it is frequently necessary to compile and link against many libraries using a cross-compiler (https://en.wikipedia.org/wiki/Cross_compiler). This is also very common for embedded systems and is also referred as tool-chains. Recall from previous workshops that widely adopted systems in robotics, such as ROS (www.ros.org) use sophisticated building tools (based on cmake named catkin.

Consider a simple C++ program (see Figure 1) under a UNIX shell (this is common in Linux, MacOS terminal or even Windows with cygwin: www.cygwin.com). This simple program Hello.cpp can easily be compiled on a terminal as follows.

```
> g++ Hello.cpp
```

This compiles and links this program to create an executable a.out. Now, to run the program, we just invoke it.

```
> ./a.out
```

It is preferable if the executable has a name that is more meaningful. This can be achieved with an option -o (usually for output):

```
> g++ Hello.cpp -o Hello
> ./Hello
```

Many programing languages build executables in at least two stages. For example, C++11, C and Objective-C first compile independent modules, producing objectfiles, and later they link them into an executable. Makefiles describe the stages to build such executables and ensure that only the necessary steps to rebuild an executable are taken when some of its components are modified. Thus, they save a lot of devleopment time when programmers are waiting for the compilation or linking process. They can also generate and execute tests after compilation and also re-generate the documentation from tools like doxygen. The utility make that uses such makefiles is independent of the compilers and the programming languages (and in fact can use entire scripts of commands to achieve a target). A simple Makefile for input to compile the program of Figure 1 is the following. Hello: Hello.cpp

```
g++ -Wall -o Hello Hello.cpp
```

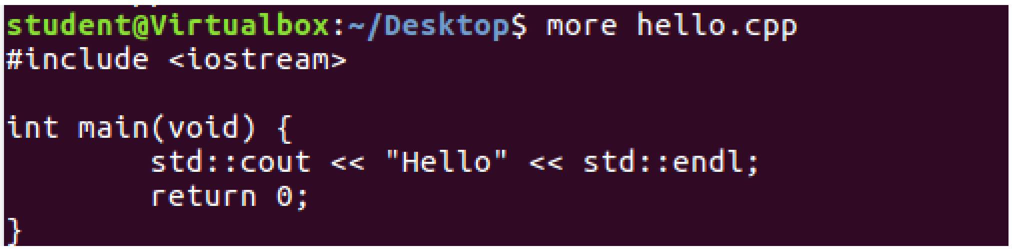
Create this file with name Makefile in the same directory. Make sure you can compile the program with the command

```
> make
```

Modify (edit) Hello.cpp to print a different message and compile it again using make. Run the executable now named Hello.

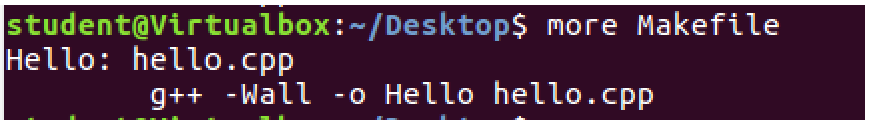
Response 1

To begin, A hello.cpp file was created which printed hello.



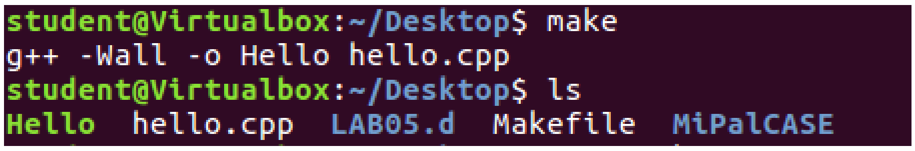
```
student@Virtualbox:~/Desktop$ more hello.cpp
#include <iostream>

int main(void) {
        std::cout << "Hello" << std::endl;
        return 0;
}
```

Alongside this, a Makefile was created to compile the hello.cpp



```
student@Virtualbox:~/Desktop$ more Makefile
Hello: hello.cpp
        g++ -Wall -o Hello hello.cpp
```

Running the command make, the file has been compiled into an executable Hello.

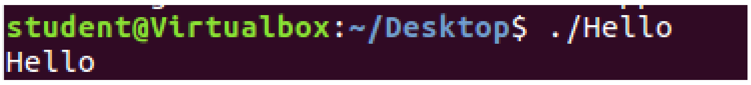


```
student@Virtualbox:~/Desktop$ make
g++ -Wall -o Hello hello.cpp
student@Virtualbox:~/Desktop$ ls
Hello  hello.cpp  LAB05.d  Makefile  MiPalCASE
```

Executing the command

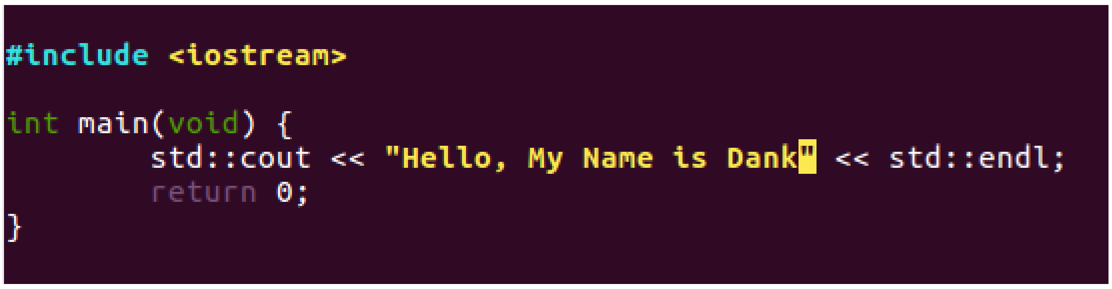
./Hello

Yielded the following results.



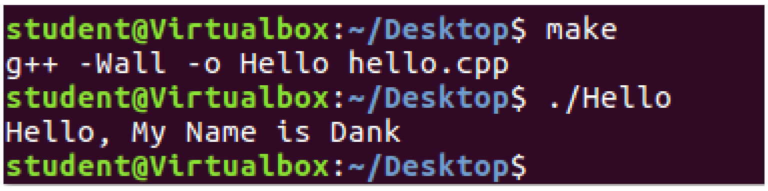
```
student@Virtualbox:~/Desktop$ ./Hello
Hello
```

The hello.cpp file was altered to change the message as follows



```
#include <iostream>

int main(void) {
        std::cout << "Hello, My Name is Dank" << std::endl;
        return 0;
}
```

Running the Makefile again will recompile the program to run the new message on execution:


```
student@Virtualbox:~/Desktop$ make
g++ -Wall -o Hello hello.cpp
student@Virtualbox:~/Desktop$ ./Hello
Hello, My Name is Dank
student@Virtualbox:~/Desktop$
```

Activity 2

Edit your Makefile and add a target for generating documentation. Namely, add the following two lines.

Documentation: Hello.cpp

doxygen

Note that the action to be performed to achieve the target is indented with a tab. Now, set the configuration of doxygen by running

> doxygen -g

This should create the Doxyfile configuration file for you. Edit this file, find the INPUT = and add Hello.cpp. Then add comments to the Hello.cpp file to create documentation as per Figure 2. Check doxygen documentaiton, you may need to add

file Hello.cpp

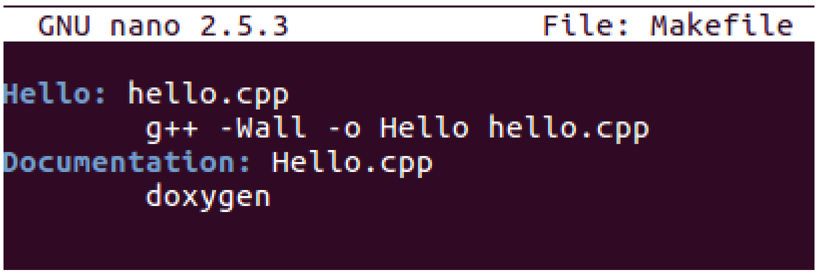
For your file to be considered. To generate the documentation, now you use make with a specific target.

> make Documentation

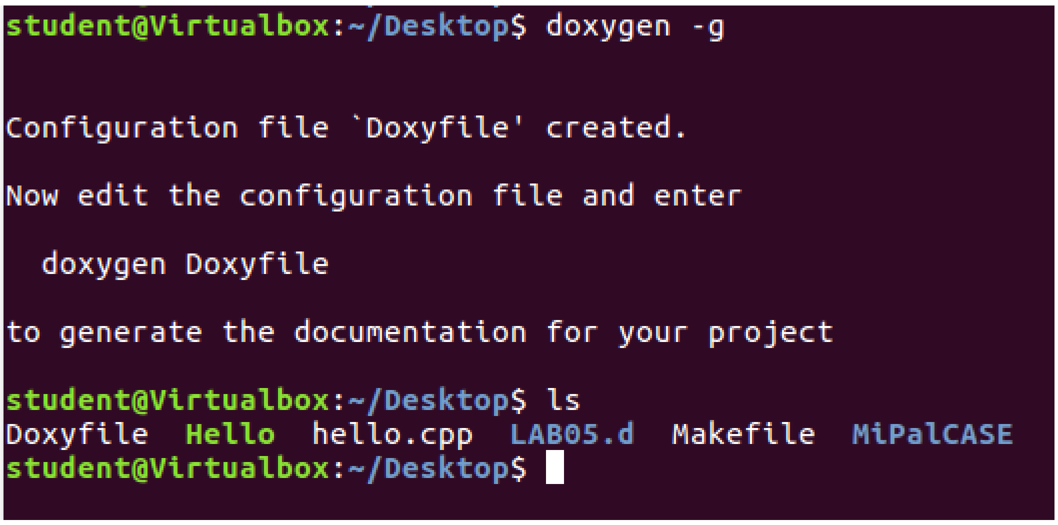
The useful doxygen tags you need here are @file, @author, @date, and @brief.

Response 2

The first step is to alter the Makefile to include the Documentation section.

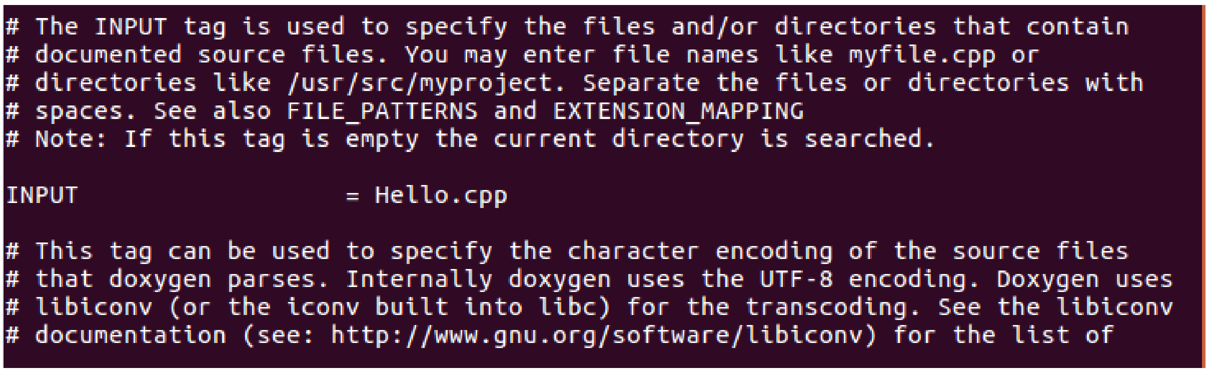

```
  GNU nano 2.5.3                File: Makefile

Hello: hello.cpp
        g++ -Wall -o Hello hello.cpp
Documentation: Hello.cpp
        doxygen
```

Using the command **sudo apt install doxygen,** the doxygen program has been installed for use. The next step is to create the Doxygen configuration file:



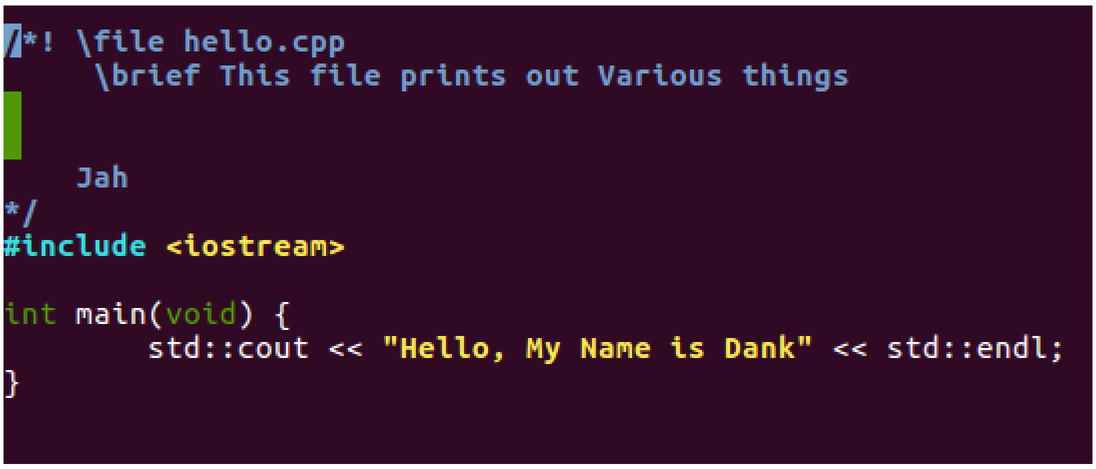
```
student@Virtualbox:~/Desktop$ doxygen -g


Configuration file `Doxyfile' created.

Now edit the configuration file and enter

  doxygen Doxyfile

to generate the documentation for your project

student@Virtualbox:~/Desktop$ ls
Doxyfile  Hello  hello.cpp  LAB05.d  Makefile  MiPalCASE
student@Virtualbox:~/Desktop$ 
```

This configuration file is then slightly altered to include the correct file Input:



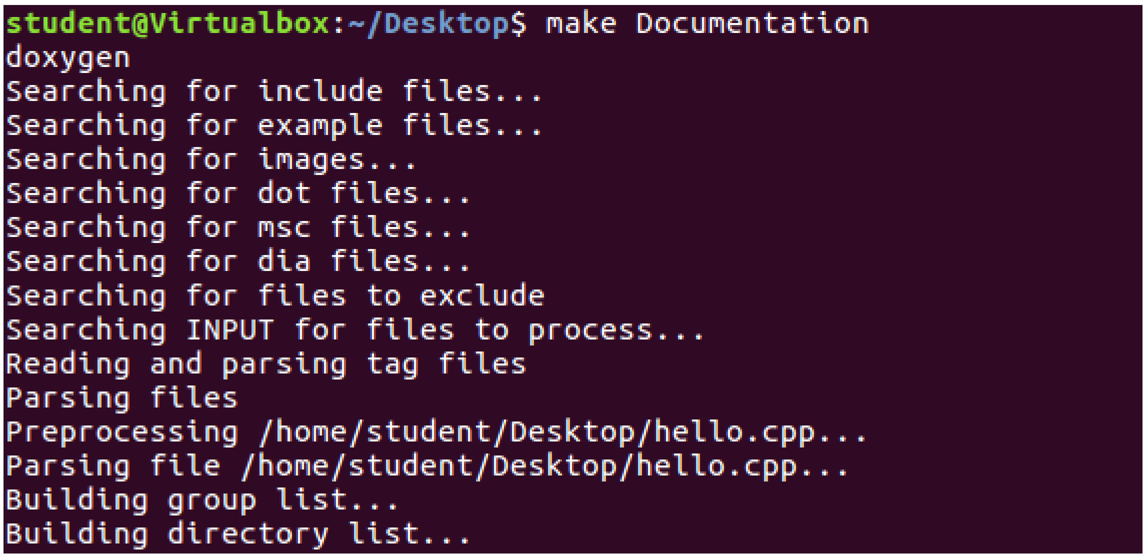
```
# The INPUT tag is used to specify the files and/or directories that contain
# documented source files. You may enter file names like myfile.cpp or
# directories like /usr/src/myproject. Separate the files or directories with
# spaces. See also FILE_PATTERNS and EXTENSION_MAPPING
# Note: If this tag is empty the current directory is searched.

INPUT                  = Hello.cpp

# This tag can be used to specify the character encoding of the source files
# that doxygen parses. Internally doxygen uses the UTF-8 encoding. Doxygen uses
# libiconv (or the iconv built into libc) for the transcoding. See the libiconv
# documentation (see: http://www.gnu.org/software/libiconv) for the list of
```

The hello.cpp file needs to be altered in order to allow doxygen to identify the file as one to be documented:

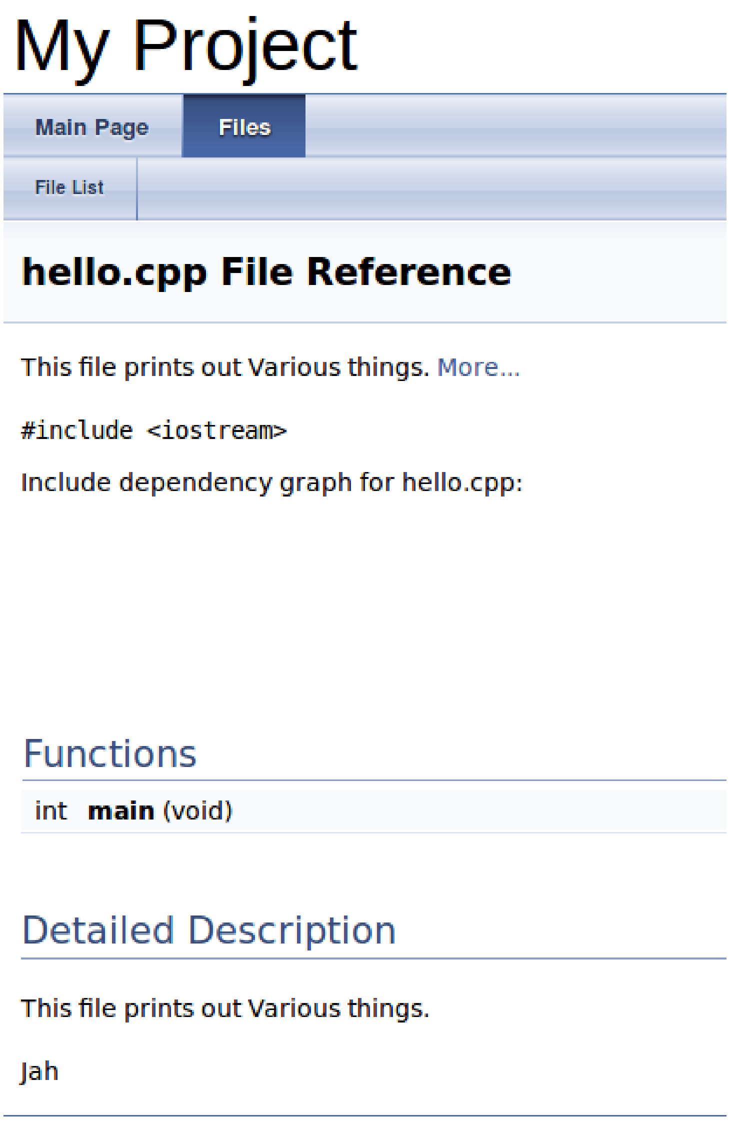


```
/*! \file hello.cpp
     \brief This file prints out Various things



    Jah
*/
#include <iostream>

int main(void) {
        std::cout << "Hello, My Name is Dank" << std::endl;
}
```

Then the command **make** must be run with the Documentation to create the doxygen file:



```
student@Virtualbox:~/Desktop$ make Documentation
doxygen
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for msc files...
Searching for dia files...
Searching for files to exclude
Searching INPUT for files to process...
Reading and parsing tag files
Parsing files
Preprocessing /home/student/Desktop/hello.cpp...
Parsing file /home/student/Desktop/hello.cpp...
Building group list...
Building directory list...
```



My Project

hello.cpp File Reference

This file prints out Various things. More...

```
#include <iostream>
```

Include dependency graph for hello.cpp:

Functions

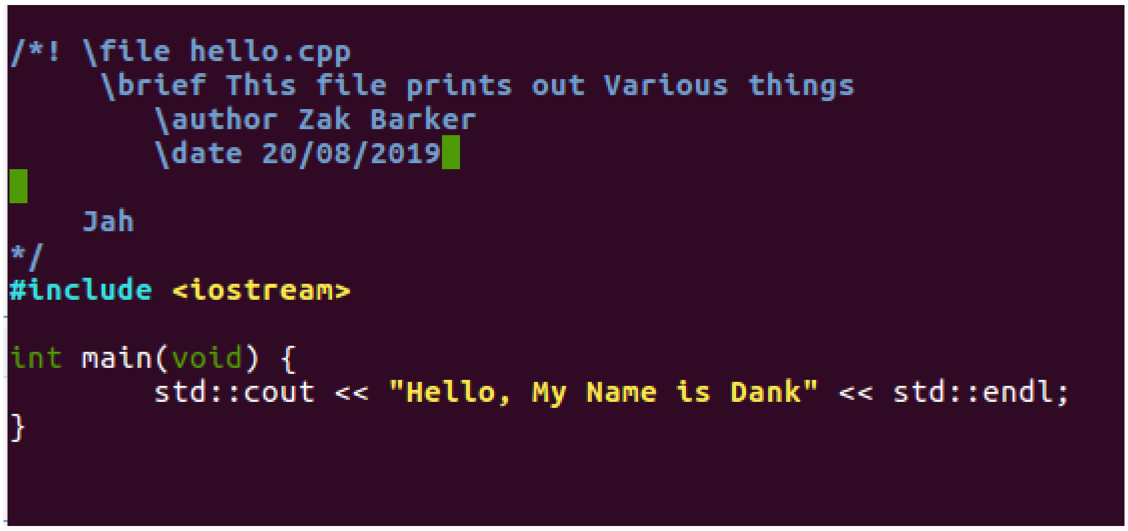
int **main** (void)

Detailed Description

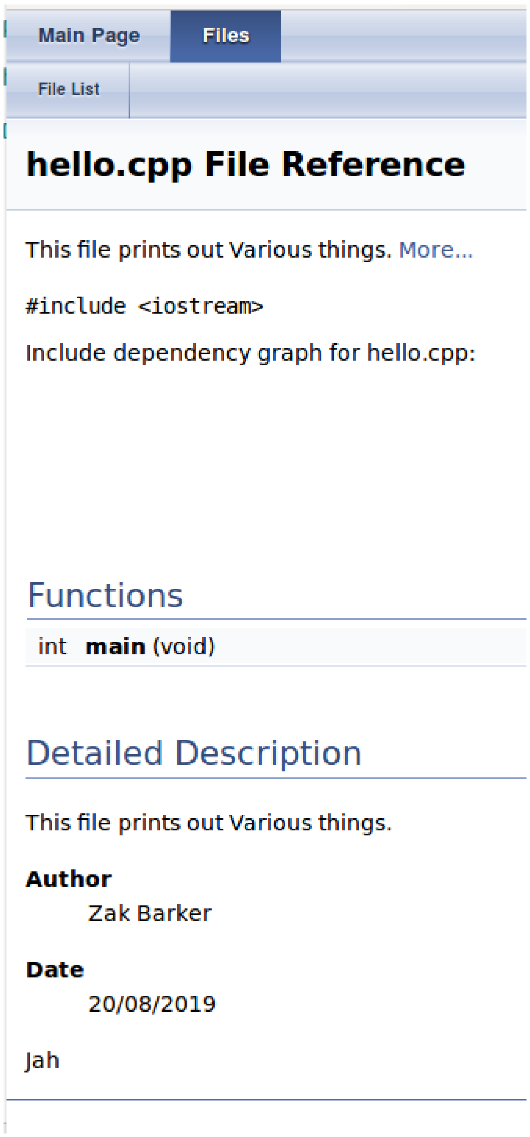
This file prints out Various things.

Jah

Finally, altering the file with /author and /date allows the completion of the form.



```
/*! \file hello.cpp
     \brief This file prints out Various things
        \author Zak Barker
        \date 20/08/2019

    Jah
*/
#include <iostream>

int main(void) {
        std::cout << "Hello, My Name is Dank" << std::endl;
}
```



hello.cpp File Reference

This file prints out Various things. More...

```
#include <iostream>
```

Include dependency graph for hello.cpp:

Functions

int **main** (void)

Detailed Description

This file prints out Various things.

Author

Zak Barker

Date

20/08/2019

Jah

Activity 3

Revisit the code provided in Workshop 3 and inspect the Makefile there. It was called Makefile2.make and to get make to use it, one needs to explicitly indicate so.

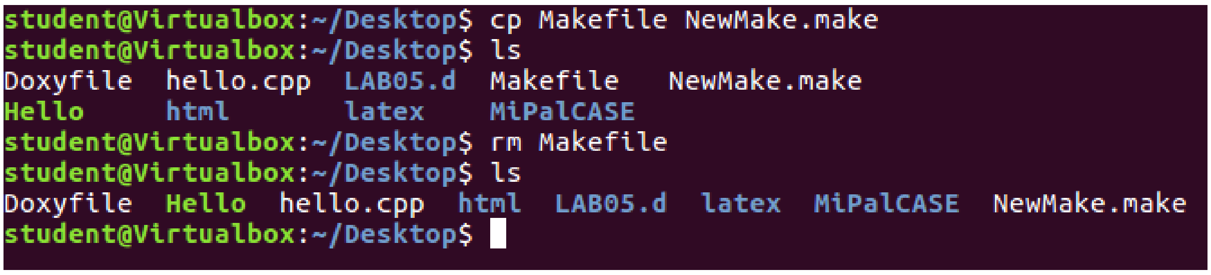
> *make -f Makefile2.make*

Make sure you can follow all the structure of this makefile. It should be possible to find in the WEB explanations for what is going on.

However, if you are curious about the complexity that makefiles can reach you are invited to visit the download page of the package PuMoC (https://lipn.univparis13.fr/ touili/PuMoC/index.html). I do not recommend you download the package, it is several Megabytes long. Note that to install this package one uses make. I would only provide the Makefile so you can observe how challenging these makefiles can get. And there are even more complicated ones.

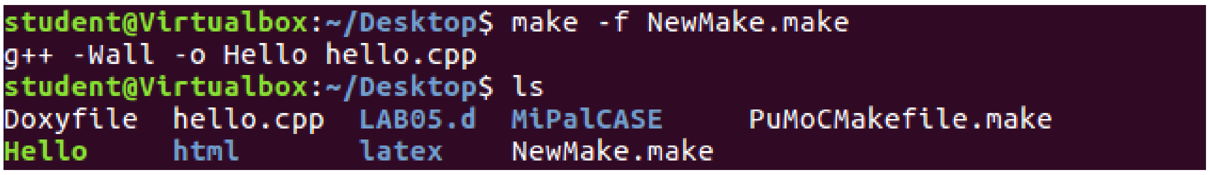
Response 3

Initially, the makefile was copied with a different name and the original makefile was deleted.



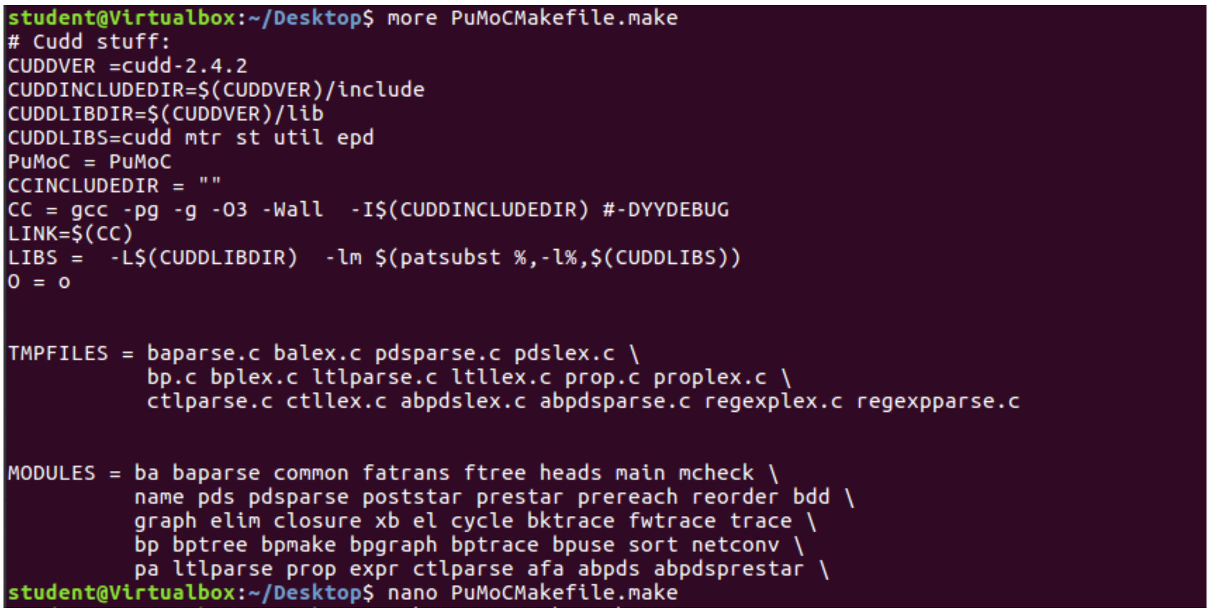
```
student@Virtualbox:~/Desktop$ cp Makefile NewMake.make
student@Virtualbox:~/Desktop$ ls
Doxyfile  hello.cpp  LAB05.d  Makefile   NewMake.make
Hello     html       latex    MiPalCASE
student@Virtualbox:~/Desktop$ rm Makefile
student@Virtualbox:~/Desktop$ ls
Doxyfile  Hello  hello.cpp  html  LAB05.d  latex  MiPalCASE  NewMake.make
student@Virtualbox:~/Desktop$ 
```

Next, the make –f command was run on the new file to ensure it runs correctly.



```
student@Virtualbox:~/Desktop$ make -f NewMake.make
g++ -Wall -o Hello hello.cpp
student@Virtualbox:~/Desktop$ ls
Doxyfile  hello.cpp  LAB05.d  MiPalCASE     PuMoCMakefile.make
Hello     html       latex    NewMake.make
```

Finally, The PuMoCMakeFile.make was run in order to investigate the complexities of Makefiles… I don’t know what to think about this currently, but one day I will be able to decipher and write files like this.



```
student@Virtualbox:~/Desktop$ more PuMoCMakefile.make
# Cudd stuff:
CUDDVER =cudd-2.4.2
CUDDINCLUDEDIR=$(CUDDVER)/include
CUDDLIBDIR=$(CUDDVER)/lib
CUDDLIBS=cudd mtr st util epd
PuMoC = PuMoC
CCINCLUDEDIR = ""
CC = gcc -pg -g -O3 -Wall  -I$(CUDDINCLUDEDIR) #-DYYDEBUG
LINK=$(CC)
LIBS =  -L$(CUDDLIBDIR)  -lm $(patsubst %,-l%,$(CUDDLIBS))
O = o


TMPFILES = baparse.c balex.c pdsparse.c pdslex.c \
           bp.c bplex.c ltlparse.c ltllex.c prop.c proplex.c \
           ctlparse.c ctllex.c abpdslex.c abpdsparse.c regexplex.c regexpparse.c


MODULES = ba baparse common fatrans ftree heads main mcheck \
          name pds pdsparse poststar prestar prereach reorder bdd \
          graph elim closure xb el cycle bktrace fwtrace trace \
          bp bptree bpmake bpgraph bptrace bpuse sort netconv \
          pa ltlparse prop expr ctlparse afa abpds abpdsprestar \
student@Virtualbox:~/Desktop$ nano PuMoCMakefile.make
```

Activity 4

In the next activity you will explore the framework JUnit to build a suite of tests for methods of a Java class. A testing framework is useful for Test-Driven Development (TDD) (you can find more information on en.wikipedia.org/wiki/Testdriven_development). Test-Driven Development typically builds the test even before the class exists. We are not going to do this here. The classes will be provided for you as well as the test, so you can become familiar with the mechanics.

Kent Beck, who popularized TDD in eXtreme Programming (XP) [1], saw many advantages. We can mention a few. The first one is that writing of the tests defines the signatures of the methods of the class. It helps you decide how you want to use the class.

The test themselves describe the behaviour of the class as input and output tests. Typically it is easier to check a result than to come up with the algorithm that produces the result. It is easier to check if the Rubik's cube is being solved with all faces of one colour that to describe how to solve any scrambled instance of Rubik's cube (see Figure 3). In fact, the fundamental question of whether the class of problems in NP is the same as the class of problems in P relates to whether checking an answer is as hard as coming up with it.

For the purposes of developing software, it is easier to get started writing the tests than the actual class (see Figure 4). It gives you a pathway to get going. Even if all tests fail at the beginning, then you can chose which test to tackle next and produce the code that pass it. Improving how many tests you pass in the next version enables a discipline of progress in your work. Note that you may discover some new tests along the way. Tests are not sufficient to establish your program is correct, but certainly, it has a big if a test fails. Read "Test Driven Development (TDD): Example Walk-through" technologyconversations.com/2013/12/20/test-driven-development-tdd-examplewalkthrough/ for an example.

Write 5 lines explaining what is the process that is suggested by the TDD method.

Response 4

The process of test driven development is to create a finished project by breaking down a complex set of requirements into bite sized sections and running small tests with the intention of failing them. With this as a starting point, the goal is to implement enough code to pass this test before refining that code and creating a new failed test. By working in small iterations with a target in mind, as the name suggests, the development of the project is driven by the process of creating tests and implementing solutions to pass them.

Activity 5

Visit the page "Writing JUnit Tests in NetBeans IDE" (netbeans.org/kb/docs/java/junitintro.html).

Complete all the exercises suggested here. However, do not wait until they ask you to run the test to actually run them, as you rarely would see the test fail.

For example, jump to the section "Running the tests", which would indicate to you where in NetBeans you request the test to run. Run them for the first time immediately after creating the tests with the assistance of the NetBeans wizard for this. Essentially, run the tests just after choosing using JUnit3.x or JUnit4.x. Because these are default created tests, they probably all fail, and also depending on your version of NetBeans, they may not look like in the tutorial. In fact, as you go trough the tutorial, some of them will be removed and new added. Every time that you modify or introduce a test, run them again.

We will provide you with the classes Vectors.java and Utils.java in Learning@GU although you can downloaded them as suggested in the tutorial. However, the subversion command did not work for me. Thus, you can skip the section "Downloading the Solution Project".

Do a bit of research and investigate what is the terminology for set up and tear down in TDD.

Response 5

The setup() function creates an environment by which a test is able to run. A test may need various parameters before it can be fulfilled and running the test itself may change the environment in which it is running. The teardown() function is used to return that environment to a clean state so that it may be rebuilt when the setup() is run again.

Activity 6

The recommended reading for this course is the book "Essential of Software Engineering" [2]. This book is not the textbook for the course because there are many topics discussed in the lectures that are not covered in the book. Nevertheless, the book has in its Chapter 1, an example of using JUnit to test a program (and its variants). The program reads some lines from a file and returns them in sorted order.

We will provide you the code from the book in Learning@GU. Use Netbeans to create a project named StringSorter. Then place the file StringSorter.java into the source files area of the project. Netbeans may point out that some include files are missing or that some type casting to String is required. You can easily use the suggestions to correct the compilation errors.

Once this class compiles you can add its test. Create the TestStringSorter.java using the method of the tutorial in the activity before, using JUnit3.x. Add the tests to the automatic test-generated file from the provided TestStringSorter.java. Add them one by one and identify why they fail, correcting the failure and setting them green before you move to the next test.

Activity 7

Write 15 lines of a reflective report on the previous activities. Analyse and evaluate the match of the activities to the learning objectives proposed in this workshop/laboratory.

Response 7

The first three activities covered the first learning objective which is about recognising the relevance of compilation and configuration of description files. Though, I already had a basic understanding of the relevance of these sorts of tools, having the opportunity to see them first hand was certainly helpful. I want to improve in all facets of programming and I have not had a lot of experience with these sorts of tools. This was an excellent introduction and something I've been needed. For that, the purpose of the first task has been met. In terms of recognizing the power of test-driven development, I feel I've scratched the surface. This is the first time hearing the term but I found the activity too difficult to follow as I'm new to Java. I do believe the activity went over my head but through research I have learnt about the concept and have at least a minor understanding of the potential of TDD. Finally, practicing with unit-testing was again, something that went over my head. Personally, I take this as an introduction to the concept and I'll be needing to improve my language skills in Java before I am able to fully immerse myself in this manner. However, now that I've heard the idea, I may be able to test out the tenets of test-driven development on other projects I have in which I am for comfortable with the language and framework. I look forward to this.