WALK-THROUGH

ONE

BUILDING C++ IN VS CODE

This walk-through is intended as a practical way to create small C++ programs on the Windows platform. *Visual Studio Code* (or *VS Code*) is a free text editor, it has some syntax highlighting capabilities and integrates with other build tools. (Using Gedit in Linux environments or XCode or Atom on Mac OS X would be similar starting-level tools; please let us know, if you plan to use non-Windows platforms to do most of your coding work.)

1.1 Objective

We need to create a few single-file programs in C++, compile and run them. We also need to run them on plaintext input files (as standard input or `STDIN`) and we also need to get plaintext output (as standard output or `STDOUT`). At the end of the coding session we want to submit all the files to a GitHub repository.

Outline of Steps

Step 1 Set up an IDE on Windows: VS Code and Microsoft C++ compiler.

Step 2 Configure the compilation task: Create the `.vscode\tasks.json` configuration file.

Step 3 Write C++ code and build: If necessary, fix compiler and linker issues.

Step 4 Run executables from terminal: Supply input and output interactively from the terminal window.

Step 5 Redirect I/O streams to files: STDIN, STDOUT, STDERR, and also display the return value of `main()`.

1.2 Steps in Detail

1.2.1 Step 1: Set up an IDE on Windows

You can follow the guide Configure VS Code for Microsoft C++.

Visual Studio Code should have its C++ plugin enabled and Microsoft Developer Tools should be installed. Go to some directory (can name it `ds-workspace` or similar). To create the initial directory structure, you can also check out code from GitHub classroom's invitation (contact instructor for details).

Open **Developer Command Prompt for VSC**. Go to the directory `ds-workspace`. Create an empty subdirectory there and open **Visual Studio Code**:

```
mkdir hello
code .
```

1.2.2 Step 2: Configure the Compilation Task

To enable easy compilation of simple one-file projects directly from Visual Studio Code, you can configure the `hello\.vscode\tasks.json` file with the following content:

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "type": "shell",
      "label": "cl.exe build active file",
      "command": "cl.exe",
      "args": [
        "/Zi",
        "/EHsc",
        "/Fe:",
        "${fileDirname}\\${fileBasenameNoExtension}.exe",
        "${file}"
      ],
      "problemMatcher": ["$msCompile"],
      "group": {
        "kind": "build",
        "isDefault": true
      }
    }
  ]
}
```

After this you can compile the file which is currently open and active in the Visual Studio Code editor by pressing three buttons: **Ctrl-Shift-B**.

You can also compile larger projects with this method, but `"${file}"` should be replaced by `"*.cpp"` in the configuration file like this:

```
"args": [
  "/Zi",
  "/EHsc",
  "/Fe:",
  "${fileDirname}\\${fileBasenameNoExtension}.exe",
  "${file}"
],
```

This method is platform-dependent and there are more robust ways to compile C++ programs, but they will be covered in subsequent walk-throughs17.

1.2.3 Step 3: Build C++ Code

Create a minimalistic C++ program. For example, name this file `hello.cpp`

```
#include <iostream>
using namespace std;
int main() {
  cout << "Print hello" << endl;
}
```

Save this file in your directory `hello`. Then press three buttons simultaneously: **Ctrl-Shift-B**. Select the first compiler (`cl.exe`) from the drop down list (or just wait until your code compiles.

1.2.4 Run Executables from Terminal

Return to the PowerShell **Developer Command Prompt for VSC**. Run the newly created executable:

```
hello.exe
```

It should print out a greeting (`"Print hello"`).

1.2.5 Redirect I/O Streams to Files

To display return code (the integer value returned by the function `main()`) you can use the following in a Windows terminal:

```
hello.exe > myoutput.txt
echo Exit Code is %errorlevel%
```

On Linux a similar code would look like this:

```
echo $?
```

Here `%errorcode%` is a special variable that contains the value of the most recent program that ran in this terminal. As the name suggests, it is not used to return any computation results, just the indication, if the process exited normally (`0` means normal or successful, any non-zero code is some sort of failed command).

1.2.6 Save Your Work to a Repository

After the coding has been successfully finished, it has to be properly saved. All grading will rely on code being in a Git repository. The suggested of actions is the following:

1. Create workspace for your code in appropriate directory. This can be done by accepting the invite link for `ds-workspace` and cloning the repository. (In other situations you may need to check out an existing project or create a new repository from scratch.)
2. Move your existing code to the new directory.
3. Reopen Visual Studio Code, run build process and tests again.
4. Update `.gitignore` to ignore dependent files.
5. Add your source files to the repository. Tag your commit and push.
6. View the repository status in GitHub Webpage.

WALK-THROUGH: SYSTEM TESTS ON VIRTUAL LINUX

Unlike the previous walk-through (preoccupied with development environment on Windows), this walk-through shows a simple production environment on a Linux machine. The Linux guest as described in this walk-through is meant to be similar to the environment where your programming tasks are graded.

Unlike Java and Scala, C++ is not meant to be a platform-independent programming technology; it may happen that it compiles and runs on one environment (such as Windows 10 using Microsoft C++ compiler), but fails to run in another environment (such as Ubuntu/Debian Linux with makefiles and the g++ compiler).

Run C++ projects in an environment with standardized operating system and its environment variables. VirtualBox and Xubuntu ensure some predicatability. Also the project files should have predictable directory layout; version control (such as GitHub) should be used during the submission. It is your responsibility to ensure that it runs on Linux - since that is the only thing that matters for the grade for programming assignments.

Note: There may be other reasonable ways how to test your code on Linux converting your laptop to Linux, using dual-boot, or using Windows Subsystem for Linux. They may be more convenient for you, if you are familiar with the technologies involved. Instructors might be unable to assist with the setup.

2.1 Objective

We need to create a minimalistic Debian-style Linux machine to be used for building and testing C++ code - using reasonable amount of automation (to integrate with the code repository and to run all testcases automatically).

2.1.1 Overview of Steps

Step 1 Install VirtualBox software: Install VirtualBox, create a guest machine slot named {tt Xubuntu} there.

Step 2 Set Xubuntu guest: Install Oracle VirtualBox and initialize its guest machine with Xubuntu OS.

Step 3 Install Basic Software on Xubuntu: C++ tools and Git.

Step 4 Run `git` from the command-line: Check out the initial repository. Commit and push your changes to the version control. Tag your commit.

Step 5 Build and run C++ code manually: Use Linux command-line (but no Makefile-related tools) to run executables from the command line.

Step 6 Build with a Makefile: Build and run your code on some testfiles using a `Makefile`.

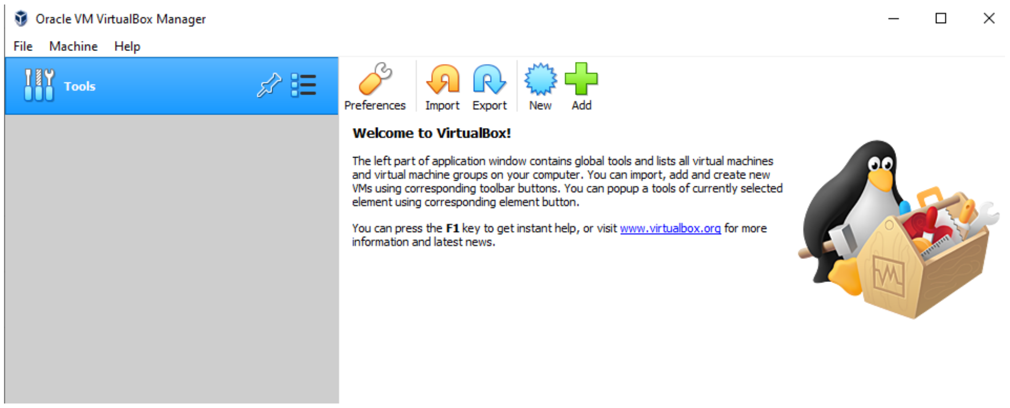
Step 7 Transfer files with `WinSCP`: Use `WinSCP`, connect to the Linux using `Putty`.

Step 8 Test out the `diff` utility: Compare the test outputs with expected outputs, use the right switches to handle Windows/Linux whitespace reasonably.

2.2 Steps in Detail

2.2.1 Step 1: Install VirtualBox software

1. Visit https://www.virtualbox.org/ and download the most recent VirtualBox installer.
   a. Click on **Download VirtualBox 6.1** banner.
   b. Click the link {bf Windows hosts}, if your physical machine is Windows 10 laptop (or choose other operating system textendash{} to whatever you have).
   c. Save the instler, such as `VirtualBox-6.1.12-139181-Win.exe`.
2. Double-click on the VirtualBox installer (elevate privileges to Admin-level, if asked to do so), and pick the default values to install it.
3. Run the newly installed application **Oracle VM VirtualBox**.

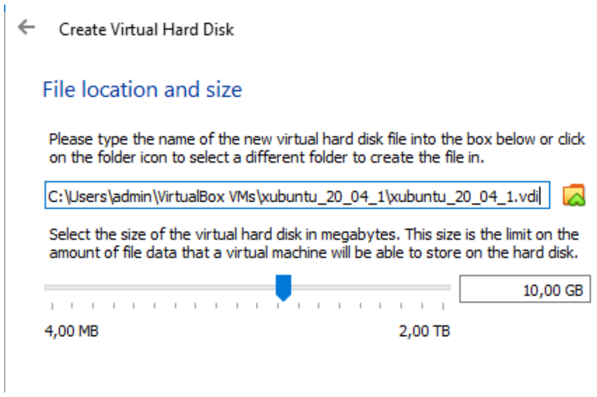


Welcome to VirtualBox!

The left part of application window contains global tools and lists all virtual machines and virtual machine groups on your computer. You can import, add and create new VMs using corresponding toolbar buttons. You can popup a tools of currently selected element using corresponding element button.

You can press the F1 key to get instant help, or visit www.virtualbox.org for more information and latest news.

4. Click button **New** and enter the name of your new virtual guest, for example, `xubuntu`.
5. Leave the default RAM memory size (1024 MiB). If your laptop is powerful (16 or more GiB of RAM), consider giving more RAM memory, say, 2048 MiB.
6. Leave the default option **Create a virtual hard disk now**; also leave the **VDI (VirtualBox Disk Image)**.
7. Leave the default option **Dynamically allocated**.
8. Confirm the location of virtual memory image.



File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

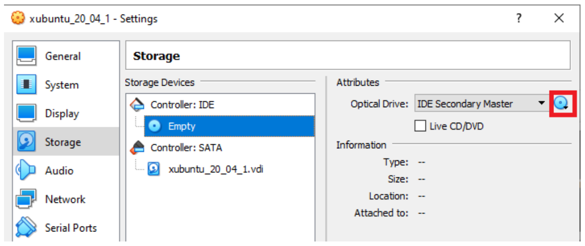
C:\Users\admin\VirtualBox VMs\xubuntu_20_04_1\xubuntu_20_04_1.vdi

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

10,00 GB

2.2.2 Step 2: Create Xubuntu Guest

1. Download the Xubuntu installer (as an ISO file of some stable release). Visit https://xubuntu.org/download/) and pick a 64-bit ISO image. In our example it is {bf xubuntu-20.04.1-desktop-amd64.iso}.
2. Make sure that the guest machine is powered off, select `xubuntu` machine and click button **Settings**.
3. Under **Settings** select **Storage** > **Controller IDE** > **Empty**.

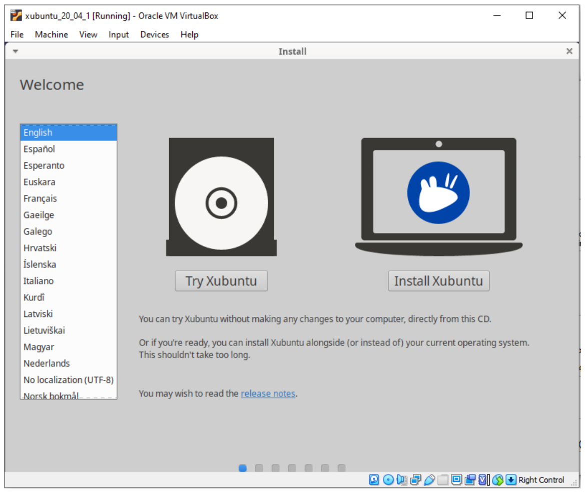


Controller: IDE
Empty
Controller: SATA
xubuntu_20_04_1.vdi

Optical Drive: IDE Secondary Master
Live CD/DVD

Type: --
Size: --
Location: --
Attached to: --

4. Click on the browse button (highlighted in red in the above image). Select the Xubuntu image that you downloaded earlier.
5. In the VirtualBox application, select the `xubuntu` machine and click on the button **Run** (the green arrow).
6. Wait about 5 minutes until Xubuntu image loads from the virtual CD-ROM drive. Click on the button **Install Xubuntu**.

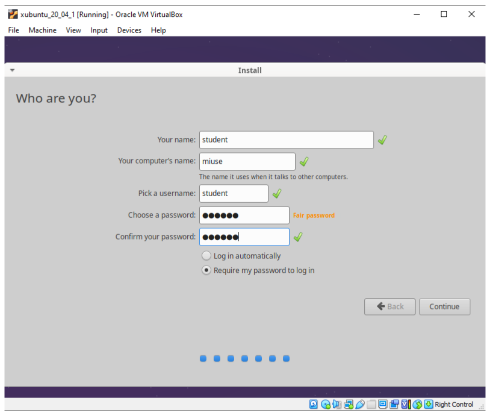


You can try Xubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Xubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

You may wish to read the release notes.

7. Leave the default keyboard layout **English (US)** > **English (US)**.
8. Selecting the checkbox **Select third party software...** in the Xubuntu installer is optional (it is selected on instructor machines).
9. Leave the default radio button **Erase disk and install Xubuntu**.
10. Select **Riga** as your current location.
11. Enter an Xubuntu Linux machine name (some short name with lower-case English letters such as `miuse`), your username (e.g. `student`) and some password (e.g. `Bitl2!`).

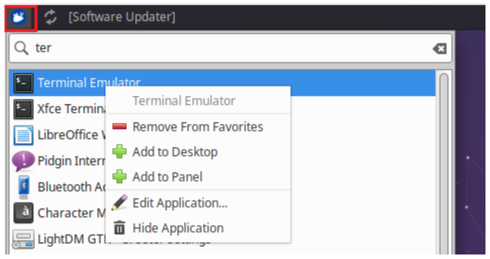

xubuntu_20_04_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Install
Back
Continue
Right Control

Who are you?

Your name: student
Your computer's name: miuse
The name it uses when it talks to other computers.
Pick a username: student
Choose a password: ●●●●●● Fair password
Confirm your password: ●●●●●●
Log in automatically
[✓ Require my password to log in]

Note: At this point you would need to wait about 15 minutes until VirtualBox finishes installing Xubuntu guest.

12. Reboot the machine. Log in as user `student` and enter the password.
13. Click on the upper-left corner (the mouse-like Xubuntu start button) and start typing word `terminal`. Once you see **Terminal Emulator**, right-click it and select **Add to Desktop**. This would make easier to create Linux-like terminal windows and run command-lines.



2.2.3 Step 3: Install Basic Software on Xubuntu

1. Set the root password to `Bitl2!` - same as for the user `student`:

```
sudo passwd
```

   First, enter `Bitl2!` password as student user. Secondly, type `Bitl2!` twice to set root's password.

2. Install all the software updates:

```
sudo apt-get update
sudo apt-get upgrade
```

3. Install C++ compiler (named `g++`) and also `make` utility:

```
sudo apt-get install build-essential
```

4. Install Git client:

```
sudo apt-get install git
```

WALK-THROUGH

THREE

WALK-THROUGH: TEST-DRIVEN DEVELOPMENT AND CATCH2

In this walk-through we have a class definition using an UML diagram and requirements for its methods. We create method stubs and testcases using the Catch2 framework. After that the methods are completed to satisfy the testcases. We also use CMake to compile the project on Windows as well as Linux.

3.1 Objective

UML diagrams are a common way to represent class design graphically. To ensure an orderly object-oriented coding we can stick to the design closely and do testcases before the code itself is written. This approach is named test-driven development; it uses the encapsulation of objects in object-oriented programming.

3.1.1 Overview of Steps

Step 1 Convert a UML diagram into method stubs: Use the given UML diagram in order to create class prototype with method stubs.

Step 2 Use Catch2 as a unit testing framework: Code testcases with requirements on behavior of class objects.

Step 3 Use CMake on Windows: Run CMake and build 2 executables – the main executable and the unit test executable.

Step 4 Use CMake on Linux: Run CMake and build the same 2 executables.

3.2 Steps in Detail

You can read Steps 1 and 2 (information on how to set up the file structure for Lab 2-1).

To demonstrate how the build process works, you can clone (or just refresh with `pull`) the GitHub project https://github.com/kapsitis/ds-workspace.git and start from Step 3.

3.2.1 Step 1: Convert a UML Diagram into Method Stubs

Read an existing UML diagram (such as Lab2-1 description). Create three files:

1. `ds-workspace-yourname\lab2-1\src\CircularList.h`
2. `ds-workspace-yourname\lab2-1\src\CircularList.cpp`
3. `ds-workspace-yourname\lab2-1\src\CircularListMain.cpp`

The header file only contains declarations of two classes `CLNode` and `CircularList`. Both of them would need to be in their namespace `ds_course` (just to avoid clashing with another `CircularList` of the same name implemented elsewhere).

Fill in all the constructors and functions from the UML class diagram (Page 2, Lab2-1 description). If the method is preceded by "+" sign, it should be public. Please note that `CLNode` declares `CircularList` as a "friend" (because `CircularList` needs to be able to write `info`, `prev`, `next` fields.

3.2.2 Step 2: Use Catch2 as a unit testing framework:

Catch2 is a unit testing framework (to automate testing the behavior of your classes and methods). Visit Catch2 project's homepage https://github.com/catchorg/Catch2. You need to have the following four files for testing:

1. `test\catch.hpp` – can be downloaded as the most recent Catch2 release.
2. `test\main.cpp` – a tiny file which serves as the entry point for unit testing.
3. `test\test_CircularList.cpp` – testcases for the class `CircularList` (provided by instructor, see testtest_circularList.cpp.
4. `cmake\Modules\ParseAndAddCatchTests.cmake` – can be downloaded from Catch2 (under subdirectory `extras`).

3.2.3 Step 3: Use CMake on Windows

Clone https://github.com/kapsitis/ds-workspace.git from GitHub (or do Steps 1 and 2 above to create your own files.)

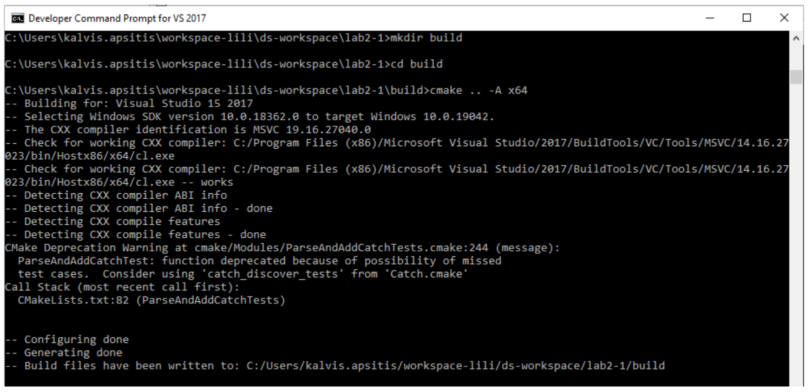
```
# Change to lab2-1 directory
cd "c:\Users\Username\ds-workspace\lab2-1"

# Create an empty "build" subdirectory (to hold compilation results)
mkdir build

# Go to that empty directory
cd build

# Create build scripts specific for the x64 architecture:
cmake .. -A x64

# Compile and link both executables:
cmake --build . --config Release
```



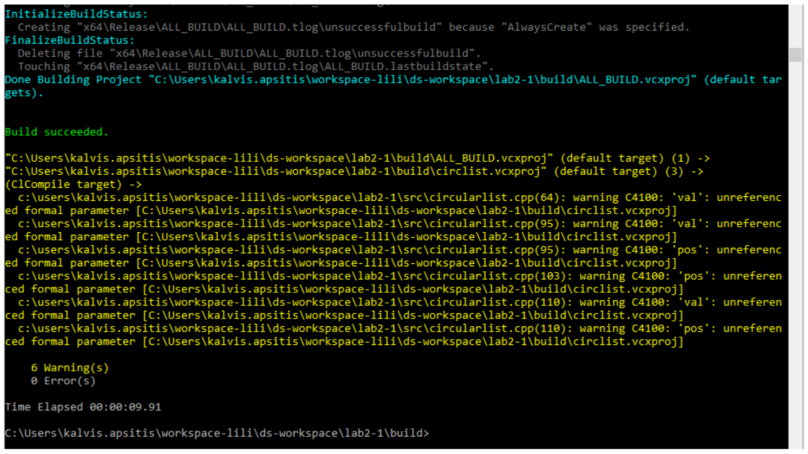
Developer Command Prompt for VS 2017

```
C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1>mkdir build

C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1>cd build

C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build>cmake .. -A x64
-- Building for: Visual Studio 15 2017
-- Selecting Windows SDK version 10.0.18362.0 to target Windows 10.0.19042.
-- The CXX compiler identification is MSVC 19.16.27040.0
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2017/BuildTools/VC/Tools/MSVC/14.16.27
023/bin/Hostx86/x64/cl.exe
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2017/BuildTools/VC/Tools/MSVC/14.16.27
023/bin/Hostx86/x64/cl.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
CMake Deprecation Warning at cmake/Modules/ParseAndAddCatchTests.cmake:244 (message):
  ParseAndAddCatchTest: function deprecated because of possibility of missed
  test cases.  Consider using 'catch_discover_tests' from 'Catch.cmake'
Call Stack (most recent call first):
  CMakeLists.txt:82 (ParseAndAddCatchTests)


-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/kalvis.apsitis/workspace-lili/ds-workspace/lab2-1/build
```



```
InitializeBuildStatus:
  Creating "x64\Release\ALL_BUILD\ALL_BUILD.tlog\unsuccessfulbuild" because "AlwaysCreate" was specified.
FinalizeBuildStatus:
  Deleting file "x64\Release\ALL_BUILD\ALL_BUILD.tlog\unsuccessfulbuild".
  Touching "x64\Release\ALL_BUILD\ALL_BUILD.tlog\ALL_BUILD.lastbuildstate".
Done Building Project "C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\ALL_BUILD.vcxproj" (default tar
gets).


Build succeeded.

"C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\ALL_BUILD.vcxproj" (default target) (1) ->
"C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj" (default target) (3) ->
(ClCompile target) ->
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(64): warning C4100: 'val': unreferenc
ed formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(95): warning C4100: 'val': unreferenc
ed formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(95): warning C4100: 'pos': unreferenc
ed formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(103): warning C4100: 'pos': unreferen
ced formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(110): warning C4100: 'val': unreferen
ced formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]
  c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\src\circularlist.cpp(110): warning C4100: 'pos': unreferen
ced formal parameter [C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\circlist.vcxproj]

    6 Warning(s)
    0 Error(s)

Time Elapsed 00:00:09.91

C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build>
```

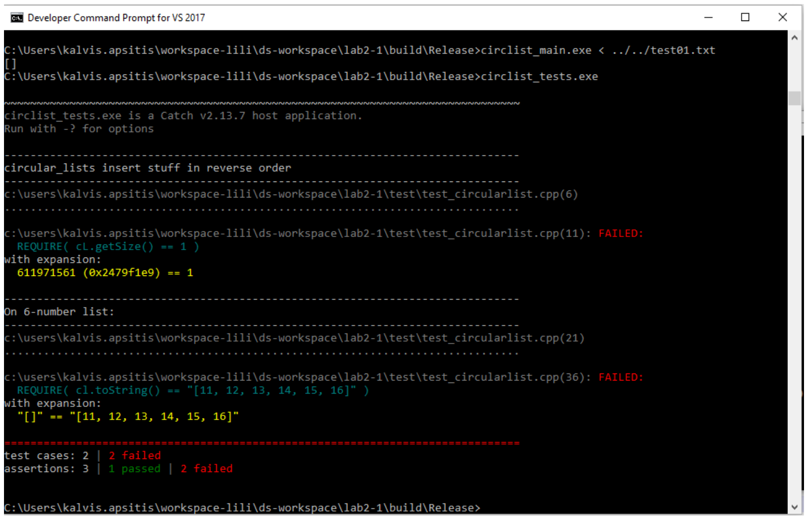
This builds two projects at the same time:

- `build\Release\circlist_main.exe` (or `circlist_main` on Linux). This is used to read plaintext files and write to output.
- `build\Release\circlist_tests.exe`. This is used to run Catch2 testcases.

You can run both of them from the Windows terminal:

```
cd Release
circlist_main.exe < ../../test01.txt

circlist_tests.exe
```



```
C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\Release>circlist_main.exe < ../../test01.txt
[]
C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\Release>circlist_tests.exe

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
circlist_tests.exe is a Catch v2.13.7 host application.
Run with -? for options

-------------------------------------------------------------------------------
circular_lists insert stuff in reverse order
-------------------------------------------------------------------------------
c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\test\test_circularlist.cpp(6)
...............................................................................

c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\test\test_circularlist.cpp(11): FAILED:
  REQUIRE( cL.getSize() == 1 )
with expansion:
  611971561 (0x2479f1e9) == 1

-------------------------------------------------------------------------------
On 6-number list:
-------------------------------------------------------------------------------
c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\test\test_circularlist.cpp(21)
...............................................................................

c:\users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\test\test_circularlist.cpp(36): FAILED:
  REQUIRE( cl.toString() == "[11, 12, 13, 14, 15, 16]" )
with expansion:
  "[]" == "[11, 12, 13, 14, 15, 16]"

===============================================================================
test cases: 2 | 2 failed
assertions: 3 | 1 passed | 2 failed


C:\Users\kalvis.apsitis\workspace-lili\ds-workspace\lab2-1\build\Release>
```

3.2.4 Step 4: Use CMake on Linux

Install cmake tool:

```
sudo apt-get install cmake
```

Clone https://github.com/kapsitis/ds-workspace.git from GitHub

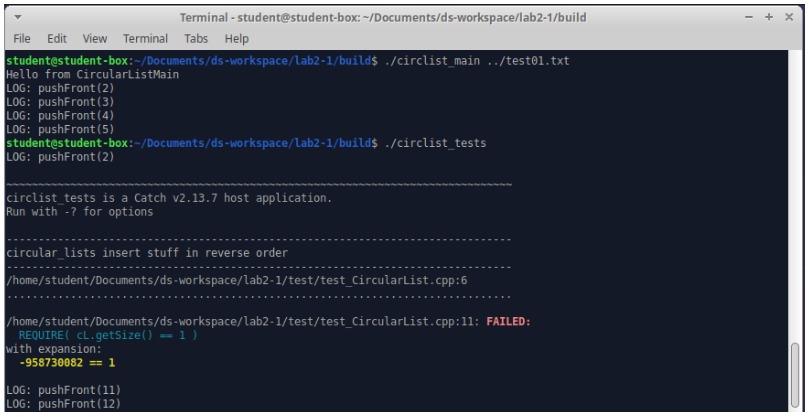
Run the following commands:

```
cd ~\Documents\ds-workspace\lab2-1
mkdir build
cd build
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Debug ..
cmake --build . --config Release
```

Run both executables `circlist-main` and `circlist-tests`:

```
./circlist_main < ../test01.txt
./circlist_tests
```

You should see messages like this:


Terminal - student@student-box: ~/Documents/ds-workspace/lab2-1/build
File Edit View Terminal Tabs Help

```
student@student-box:~/Documents/ds-workspace/lab2-1/build$ ./circlist_main ../test01.txt
Hello from CircularListMain
LOG: pushFront(2)
LOG: pushFront(3)
LOG: pushFront(4)
LOG: pushFront(5)
student@student-box:~/Documents/ds-workspace/lab2-1/build$ ./circlist_tests
LOG: pushFront(2)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
circlist_tests is a Catch v2.13.7 host application.
Run with -? for options

-------------------------------------------------------------------------------
circular_lists insert stuff in reverse order
-------------------------------------------------------------------------------
/home/student/Documents/ds-workspace/lab2-1/test/test_CircularList.cpp:6
...............................................................................

/home/student/Documents/ds-workspace/lab2-1/test/test_CircularList.cpp:11: FAILED:
  REQUIRE( cL.getSize() == 1 )
with expansion:
  -958730082 == 1

LOG: pushFront(11)
LOG: pushFront(12)
```

Currently `circlist_main` is only printing a greeting (plus redundant log messages). And `circlist_tests` is failing all the tests. This is expected: Both executables built successfully, but the unit testing failed, since nothing meaningful has been implemented and the actual outputs from the class `CircularList` methods do not match the expected results.