First of all, having clang-2.8 and llvm-2.8 is extremely necessary.

Follow the steps for installing both. Please copy the commands correctly

Download the following zip file:

https://releases.llvm.org/2.8/llvm-2.8.tgz

Extract this.

$ cd llvm-2.8/tools

$ wget http://llvm.org/releases/2.8/clang-2.8.tgz

$ tar -xzf clang-2.8.tgz

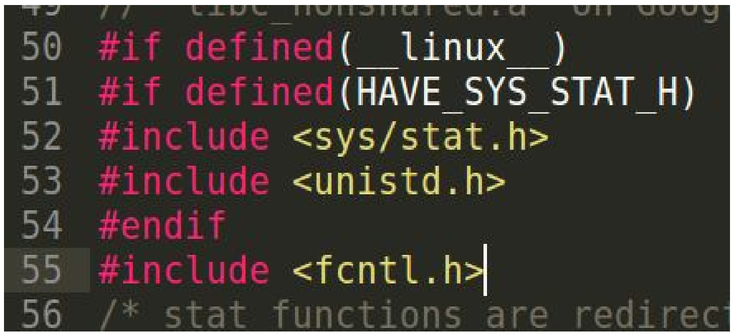
$ mv clang-2.8 clang

$ cd ../../

Now there are two changes which need to be done since this is an older clang and llvm version.

Change 1:
Go to the file llvm-2.8/lib/ExecutionEngine/Intercept.cpp

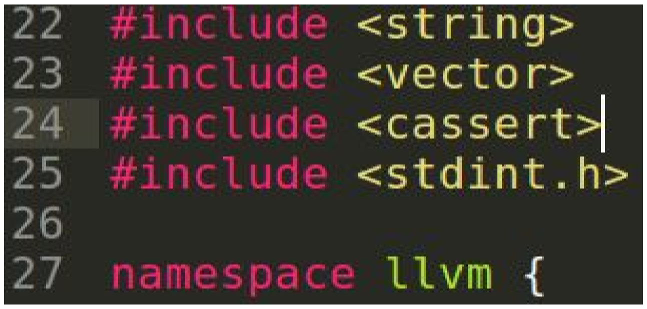
Add #include <unistd.h> after between original line 52 and line 53. It should look like this



```
49  // libc_nonshared.a on Goog
50  #if defined(__linux__)
51  #if defined(HAVE_SYS_STAT_H)
52  #include <sys/stat.h>
53  #include <unistd.h>
54  #endif
55  #include <fcntl.h>
56  /* stat functions are redirect
```

Change 2:
Go to the file llvm-2.8/tools/clang/include/clang/Diagnostic.h

Add #include <stdint.h> after line 24. It should look like this.



```
22  #include <string>
23  #include <vector>
24  #include <cassert>
25  #include <stdint.h>
26
27  namespace llvm {
```

Now go ahead with these commands.

```
$ mkdir mybuilddir
```

```
$ cd mybuilddir
```

```
$ cmake path/to/llvm/source/root
```

Note: Here you have to put the path of your llvm folder. Eg. cmake /home/puneet/llvm-2.8

$ sudo `cmake --build .`

$ sudo `cmake --build . --target install`

```
Clang 2.8 and llvm-2.8 are installed
```

References:
https://llvm.org/docs/CMake.html
http://left404.com/2012/10/17/building-clang-on-ubuntu-linux-12-04/

```
Now make sure that the 2.8 versions are installed.

$ llvm-config --version

This should show 2.8 in terminal

$ clang --version

This should also show like the following:

clang version 2.8 (branches/release_28)
Target: x86_64-unknown-linux-gnu
Thread model: posix

It is ok if the red text doesn't match.
```

Ensure there is no error after every command and dont go ahead if there is one.

Run these commands.

$ sudo apt-get install g++-multilib
$ sudo apt-get install gcc-multilib

Now all setup is done. We can go for running the c_example and aa_example

Please study the makefile. Atleast the first 20 lines which give info about commands to be run.

*Before starting, do "**source ahir_bashrc**" in the **release folder only** and not any other since the ahir_bashrc takes the location of the folder you are running the source command in as AHIR folder. After doing this, you can go to wherever you have kept the example code folders (using the same terminal).*

*Let's start with **aa_example:***

There are 4 basic commands:

make aalink
make aa2vc

make vc2vhdl
make vhdlsim
Now various combinations of these are given in the Makefile. I would suggest atleast for some days till you get used to, run them independently so you can look at errors separately for each one.

***make aalink** is very important since it compiles you Aa code. Please run this independently. The errors (if there are) are listed somewhere in the terminal output of this command. You can guess a error if there is following line at end:*

Info: added 0 bits of buffering during path balancing.

*It is possible error is not there if this happens (less probable). Anyways, please look at the terminal output for an error when you run **aalink**.*

*So after vhdlsim, you can either use **tmux** or **Terminator**(a software :p)*
Using Terminator, it is easy to split terminals and look at both terminals simultaneously.
***Note:** But don't forget to source ahir_bashrc in both terminals.*

After running ./testbench_hw and ./ahir_system_test_bench parallely, you should get the expected output like sir showed in the demo in the last lecture.

For making gtkwave file, do ./ahir_system_test_bench --wave=name.ghw
You can also add some stop time on your own or else the gtkwave file generation will stop after you exit using Ctrl+C (which is also ok since you don't always know the expected end time).
./ahir_system_test_bench --stop-time=1500ns --wave=name.ghw

This was for HW verification.

For software verification, you can run make SW and generate testbench_sw directly but I would recommend to run make aalink separately first to see compilation errors and then make SW.

For software verification, run ./testbench_sw and see terminal output. You need to understand the testbench to understand what exactly it is printing.

*Now let's go to **c_example**:*

I would recommend doing make TOAA first and check for some errors instead of going right ahead to make HW.

You might get some error that some header file is missing. You can search about that on the internet or contact me. Hopefully you won’t get any since we already installed g++-multilib before starting.

Here too software verification can be done using make SW (no need to run anything separately since you are starting with C file itself).