**Build Linux Programmable Libraries, Makefiles,**

**Memory Management, Compilation & Linking,**

**C Programming**

I. Course: For Developers - Build Linux Programmable Libraries, Makefiles, Memory Management, Compilation & Linking, C Programming

Teaching about?:

How to develop Linux System C/C++ Libraries which are:

> Generic

> Extensible

> Programmable

> Modulazied

If we have to become programmer (in any area), you will write most of the code using libraries already created for you by your ancestors

After doing this course, You should be able to write a reusable code as a library and reuse it in applications

We will take the example to build a library of "Double Linked list" throughout the course, the techniques learnt can be applied to build library for any other data structure

It is expected that you are aware of C programming already and knows how doubly linked list words

II. What is a library?

In Simple words, a library is a reusable code that can be integrated with any application and hence, application can use it

For example, LinkedList, Stacks, Queues, Trees, Graphs exists as libraries because they can be reused again an again by various applications per the requirement

All languages have their standard set of libraries which developers uses all the time while writing code

> memcpy, memset, strcpy, malloc, free are all examples of functions defined in GNU C standard library glibc

III. Compile Commands:

Compile the application.c:

gcc -g -c application.c -o application.o

Compile the dll.c

gcc -g -c dll.c -o dll.o

Linking and create final executable:

gcc -g application.o dll.o -o exe

Run the executable:

./exe

Note:

gcc - Compiler

-g - GDB Flags (Later)

-c - compile

-o - output file

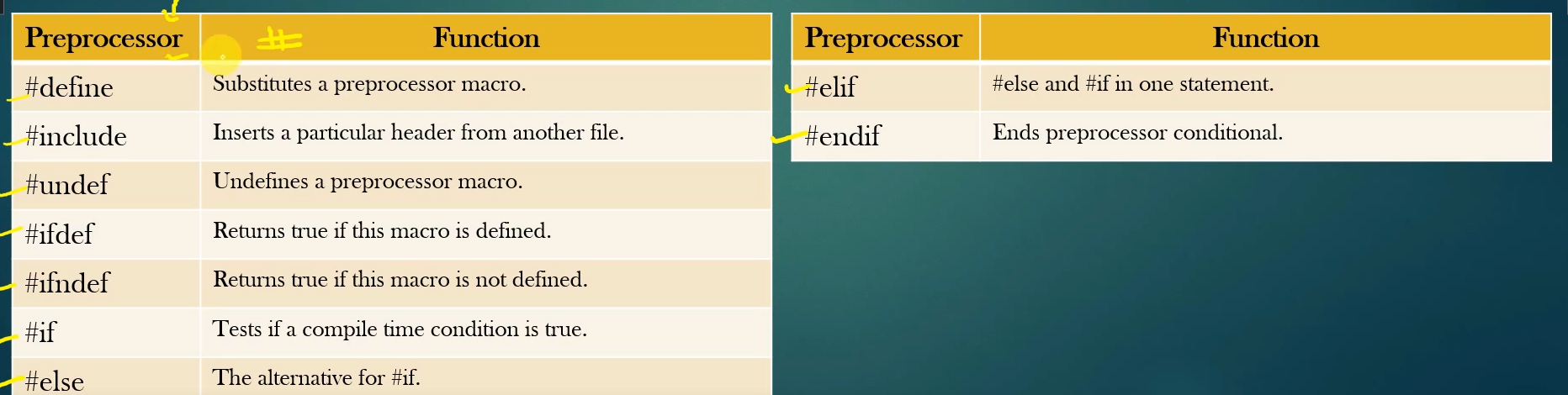
IV. Pre-Processor Directives in C

> Before compiler actually compile the source files, it performs the text substitution

> The C preprocessor directives (#include, #define) is just a simple text substitution tool

> Remember, Text substitution is performed first before the compiler actually start compilation of source files.

> Directives can be written in both - Source file as well as Header files

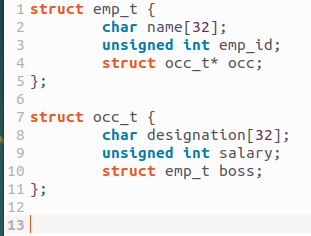


V. Way of using structure/Functions:

> Never use before defining it

> Always declarations first and then usage – for functions

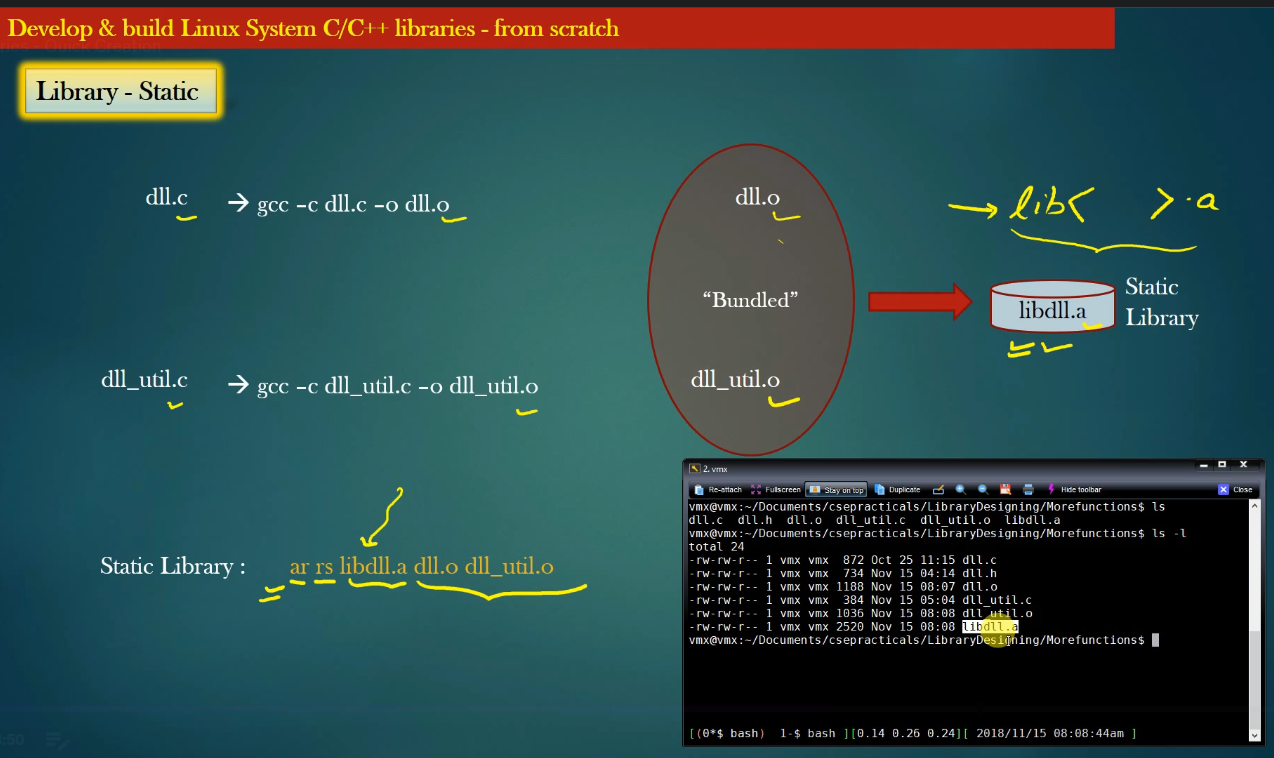
> Forward declarations for solving recursive dependency



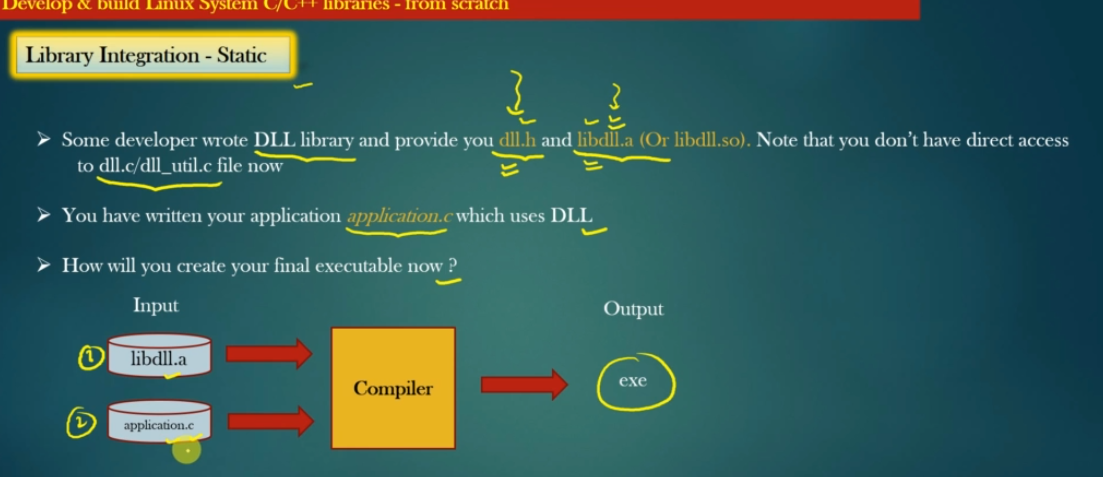
*Recursive Dependency Error*

VI. Create static and dynamic libraries:

a. Static Library



Linking with static library



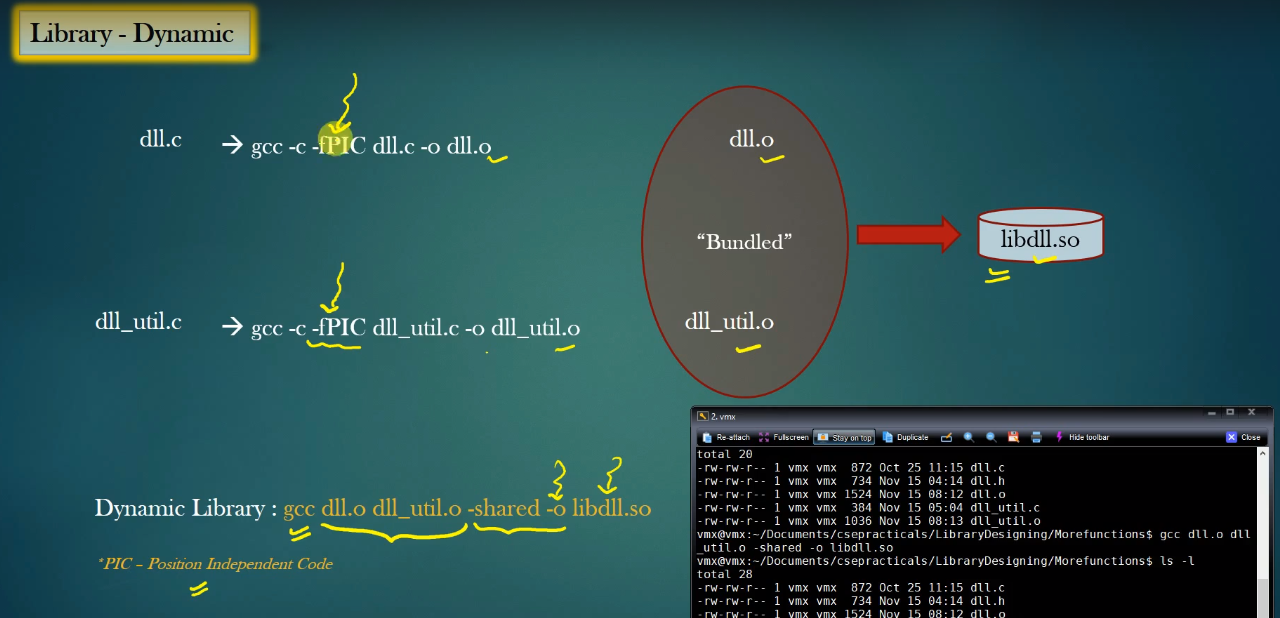
Steps:

1. gcc -c application.c -o application.o

2. gcc application.o -o exe -L . -ldll

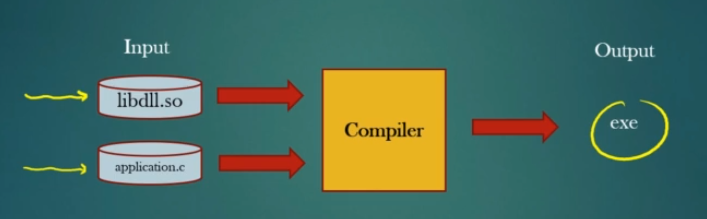
3. Run executable: ./exe

b. Dynamic Libraries



Create a dynamic library

Linking:



Steps:

gcc -o application.c -o application.so

Place the libdll.so file in the default location in */usr/lib and run sudo ldconfig command*

*gcc application.o -o exe -ldll*

*Run executable: ./exe*

VII. Four stages of compilation

Tex substitution

Execute Preprocessoing directives

Generation assbemly code

Generate machine code (0/1)

From compiled code

Libraries

Final Executable

Linking with dependent Libraries and Other Object files

VIII. Makefiles:

> Makefile is a program building tool runs on Unix, Linux, and their flavor

> it aids in simplifying building the software program that may dependent on various other libraries

> Functions of make file:

+ Compiling

Linking

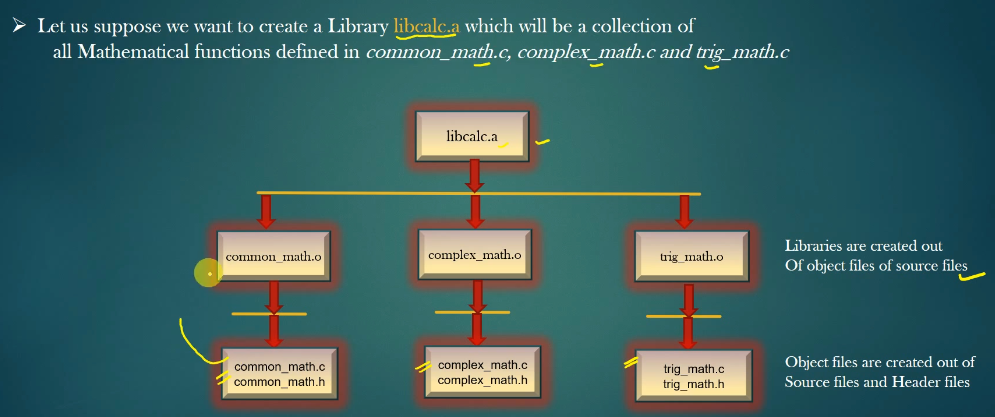
Creating required libraries – static and dynamic

Create required Executable

Installation of Libraries & executable

Update dependent

Dependency Tree:



Rules for writing make files:

<What we want to prepare (Final Dish)> : <What are raw materials we need to prepare to the final dish>

<Action – Step to prepare>

**Example**:

common\_math.o : common\_math/common\_math.c

gcc -c -I common\_math common\_math/common\_math.c -o common\_math/common\_math.o

Note:

-I <path> is used to specify the location of header files

Do the same for 2 another folder of sources

**Building a libcal.a:**

libcal.a: trig\_math/trig\_math.o common\_math/common\_math.o complex\_math/complex\_math.o

ar rs mylibmath.a trig\_math/trig\_math.o common\_math/common\_math.o complex\_math/complex\_math.o

exe: main.o libcal.a

gcc main.o –o exe -L . libcal.a -lm

Note: -lm → libmath.so

main.o : main.c

gcc -c -I common\_math -I complex\_math -I trig\_math main.c -o main.o