This lecture is regarding External Library

Static libraries
1, Combine multiple object files into a single archive file (file extension “.a”)
2, Example: libc.a contains a whole slew of object files bundled together.

To create a static library:
```
ar rs list.a List.o student.o typeUtil.o
```
We create a static library list.a, bundles three .o files.

You can check the list of .o files in a static library by using:
```
ar t list.a
```

the C standard library
```
ar -t /usr/lib/x86_64-linux-gnu/libc.a | sort | less
```

the C math library
```
ar -t /usr/lib/x86_64-linux-gnu/libm.a | sort | less
```

To use the static library, see demo in folder staticlib
Linker can also take archive files as input.

```
1, gcc -c main.c
   this compile main.c into main.o file
2, gcc -o glist main.o ./list.a
   link main.o against list.a library
3, run the executable created in step two.
   ./glist
```

Important Notes about the linking order:
1, Command line order matters!
2, Moral: put libraries at the end of the command line.
Why?
Scan .o files and .a files in command line order.
During the scan, keep a list of the current unresolved references.
As each new .o or .a file is encountered, try to resolve each unresolved reference in the list against the symbols in the new file.
If any entries unresolved when done, then return an error.

Disadvantages:
Static libraries have the following disadvantages:
1, Lots of code duplication in the resulting executable files
2, Every C program needs the standard C library.
	e.g., Every program calling printf() would have a copy of the printf() code in the executable. Very wasteful!
3, Lots of code duplication in the memory image of each running program
	OS would have to allocate memory for the standard C library routines being used by every running program!
4, Any changes to system libraries would require relinking every binary!

==
Better Solution: Shared libraries
Libraries that are linked into an application dynamically, when a program runs!
1, On UNIX, “.so” filename extension is used
2, On Windows, “.dll” filename extension is used

How it works?
When the OS runs a program, it checks whether the executable was linked against any shared library (.so) files.
	-If so, it performs the linking and loading of the shared libraries on the fly.

-The executable only keeps a small table of references to functions defined in the dynamic library,
-instead of copying the relevant machine code into the executable.

How to create? Demo

1, compile your source using

```
        gcc -c -fpic student.c
        gcc -c -fpic List.c
        ………
```

-fpic is required, called position-independent code(PIC).

2, gcc -shared -o list.so List.o student.o typeUtil.o
we created list.so (shared object) a dynamic library.

More information can be found here:
http://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html

How to use .so library? Part one

1, compile your main, in which you call functions defined in the shared library.

```
        gcc -c main.c
```

2, Link your main agaist the shared object (the external library)

```
        gcc -o glist main.o ./list.so
```

-you can create the final executable glist by using the library.

3, run program

```
        ./glist
```

How to use .so library? Part two

In real world applications, after the developer installed an external library, the .so files and .h files are placed somewhere, depending on your user privilege.

-Case one, if you install the external library as root, the .so files and .h files go to system-wide folders,
such as /usr/include for .h files, and /usr/lib for .so files
In this case, you can use the installed library just the same way as you use the C standard library.

-Case two, if you install the external library as normal user, the .so file and .h files go to a local folder,
Demo
All my .h files are placed in ~/localroot/include/ncurses
All my .so files are placed in ~/localroot/lib

Then in my .bashrc file, I add the following:

```
#####-----------------------------------------------------------------
export C_INCLUDE_PATH=/home/EASTERN/ytian/localroot/include/ncurses:/home/EASTERN/ytian/localroot/include
export LIBRARY_PATH=/home/EASTERN/ytian/localroot/lib
export LD_LIBRARY_PATH=/home/EASTERN/ytian/localroot/lib

export PATH="$PATH":/home/EASTERN/ytian/localroot
#####------------------------------------------------------------------------
```

Then I compile my main.c by using gcc -c main.c
Then linking,

```
        gcc -o glist main.o -llist
```

Note here, the dynamic library for the list are renamed to liblist.so before I copied into ~/localroot/lib
Then run your program: ./glist