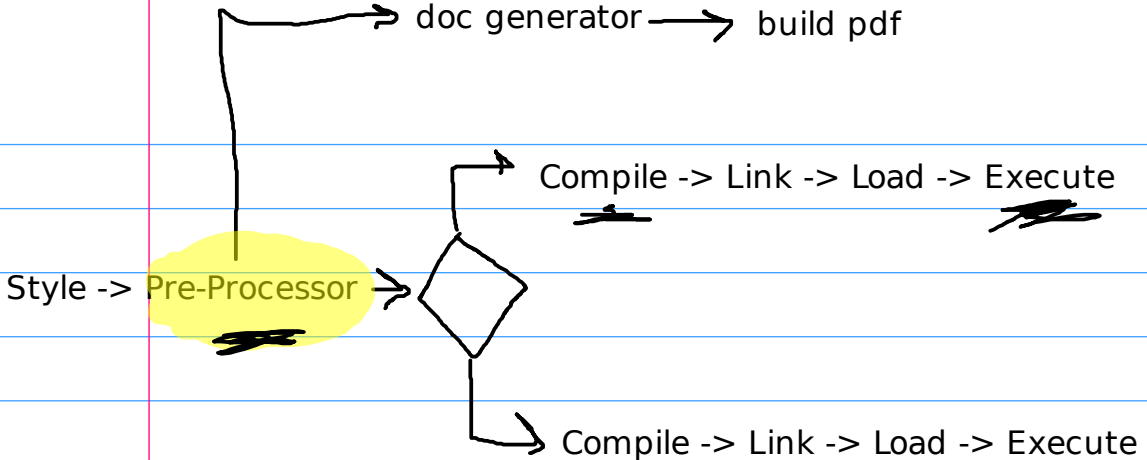
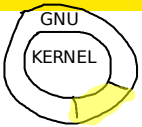
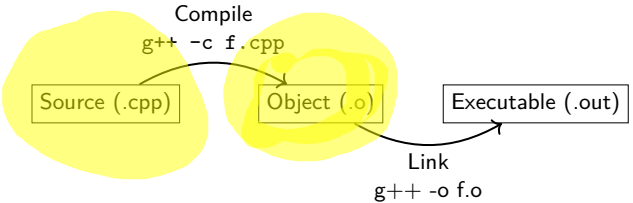

doc generator
build pdf
Compile -> Link -> Load -> Execute
Style -> Pre-Processor
Compile -> Link -> Load -> Execute

Build Pipelines





C/C++
-> CLANG
-> CLANG++

GNU

```
g++ [options] source.cpp
```

```
gcc [options] source.c
```

```
clang++ [options] source.cpp
```

```
clang [options] source.c
```

```
icpc [options] source.cpp
```

```
icc [options] source.c
```

Intel

Microsoft Visual Studio

Sources

Emacs, Vim <- Shell
Nano <- Shell

MVstudio

Integrated Development Environment

- CodeBlocks, Vim, Sublime Text, IntelliJ, Eclipse, Xcode, Android Studio, Geany, pyCharm, ... [a]
- Whichever you are comfortable with.
- Desirable Features:
  - Auto Complete features.
  - Auto Compilation features.
  - Debugger integration.
  - Code folding.
  - One IDE for all (development, office work)
  - Cross-Sectional Editing.
  - Remote development.
  - Integration with versioning systems
  - ...

Spyder

Co-Pilots (code generation, co

[a] pypl.github.io for ranking

Sources (cont.)

Naming and Style Conventions

- cpp, hpp in C++, and c, h in C.
- followCamelCaseNamingConvention
- Folder conventions: *src* for source code, *build* or *bin* for binaries, *lib* for libraries, *doc* for documentation, *test* for unit testing.
- Suffixes: *g_* for global, *s_* for static, *c_* for constant variables.
- ... Much more on **google.github.io/styleguide** (Note: Company specific Guides)
- Can be enforced at commit time, or using static code checkers (e.g. **cpplint**)
- Indentation can be forced using **indent** with args gnu (for Stallman), kr (for Kernighan & Ritchie), linux (for Torvalds), orig (for Berkeley). Check man pages for details.

Sources (cont.)

Comments -> Documentation Generator -> User Docs

User Manual

API Website

Comments

- Governed by style guidelines.
- Documentation generators (Latex, HTML, XML, PDF, Man): Doxygen, Sphinx
- Doxygen Step 1: Fix settings
- Doxygen Step 2: Write comments (E.g. below)

```
/**
 * Search whether a given directed edge exists in a graph. Edges are specified as
 * head-tail pairs and must be specified in order.
 * @param G Graph object
 * @param head First incident vertex on edge
 * @param tail Second incident vertex on edge
 * @return Not Found = 0, Found = 1
 */
int directedEdgeExists(struct nGraph *G, int head, int tail) { }
```

- Doxygen Step 3: Generate documentations

Pre-Processing iostream.h -> OS environment var

#include <iostream>
#include "hello.h"

- Source code transformation by Pre-Processor
- Pre-processor behavior controlled by directives
- File Inclusions:

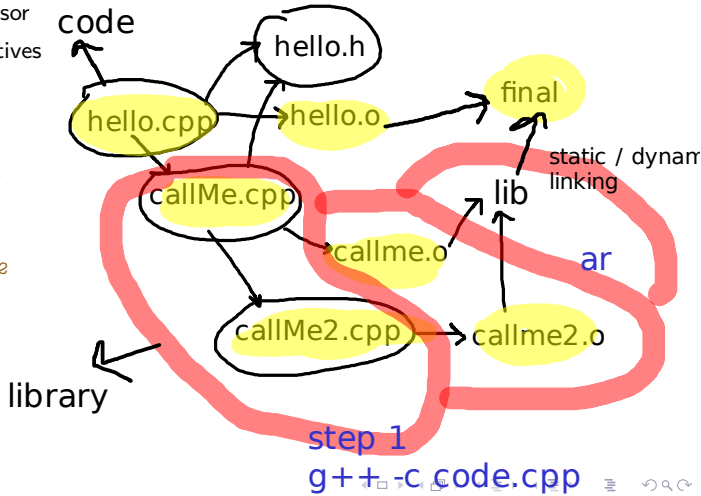
```
#include <iostream>
#include <boost/tokenizer.hpp>
#include "my_header_file.hpp"
#include "some_directory/my_header_file.hpp"
```

- Macro definitions

```
#define DEBUG_LEVEL 10
#define myMacro(param1, param2) param1+param2
```

- Conditional Compilation Directives

```
#if, #ifdef, or #ifndef directive
#elif
#else
#endif
```



Compilations

Common Arguments

- -c Compile only
- -g Debugging symbols
- -p Performance symbols
- -O*n* Optimization level to *n* (0; none, 3: full)
- -stc Version (e.g. -std=c++20)
- -I Include directory
- -L Library path
- -l*txt* Linking with library (lib*txt*) file
- -Wall enable most warnings
- -Wextra enable extra warnings
- -Werror treat warnings as errors

filesize, speed

-std

Compilations (cont.)

ELF64

- Executable and Linkable Format (for all executables, object code, libraries)
- Described in elf.h of Linux kernel
- Learning Benefits: Digital forensics, OS internals, Malware research
- Viewable using `readelf`

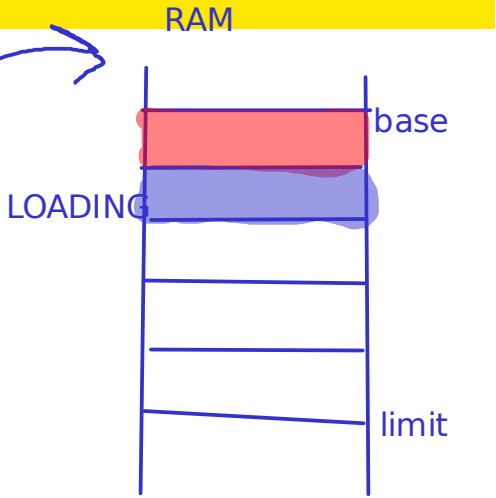
ELF File Structure

- ELF Header (64 bytes, -h switch)
- Program Header (-l switch)
- Section Header (-S switch)

ELF File Types

- Relocatable
- Executable
- Shared Object Files

```
Section Headers:
  [Nr] Name              Type            Address           Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS        0000000000000000  00000040
       000000000000003b  0000000000000000  AX       0     0     1
  [ 2] .rela.text        RELA            0000000000000000  000003f0
       0000000000000090  0000000000000018   I      11     1     8
  [ 3] .data             PROGBITS        0000000000000000  0000007b
       0000000000000000  0000000000000000  WA       0     0     1
  [ 4] .bss              NOBITS          0000000000000000  0000007b
       0000000000000000  0000000000000000  WA       0     0     1
  [ 5] .rodata           PROGBITS        0000000000000000  0000007b
       000000000000000f  0000000000000000   A       0     0     1
  [ 6] .comment          PROGBITS        0000000000000000  0000008a
       000000000000001c  0000000000000001  MS       0     0     1
  [ 7] .note.GNU-stack   PROGBITS        0000000000000000  000000a6
       0000000000000000  0000000000000000           0     0     1
  [ 8] .note.gnu.pr[...] NOTE            0000000000000000  000000a8
       0000000000000030  0000000000000000   A       0     0     8
  [ 9] .eh_frame         PROGBITS        0000000000000000  000000d8
       0000000000000038  0000000000000000   A       0     0     8
  [10] .rela.eh_frame    RELA            0000000000000000  00000480
       0000000000000018  0000000000000018   I      11     9     8
  [11] .symtab           SYMTAB          0000000000000000  00000110
```

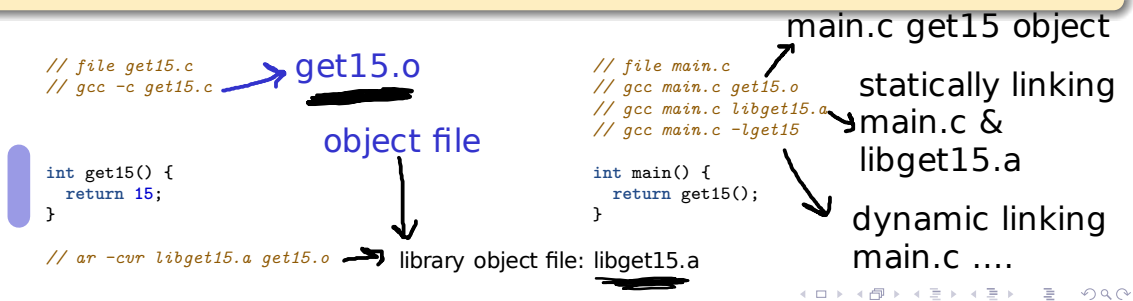


delf -h obj.o

Linking Process

Static Linking

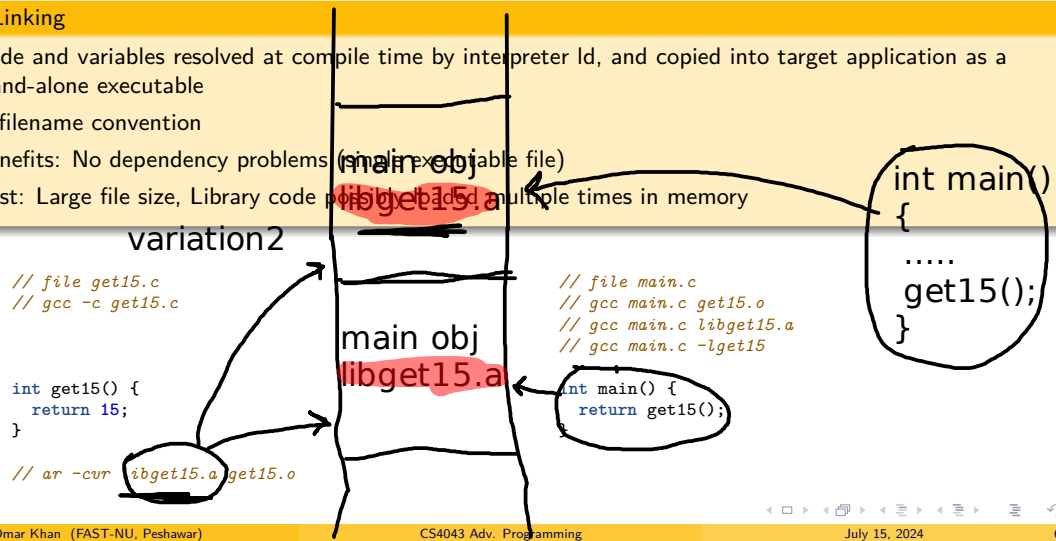
- Code and variables resolved at compile time by interpreter ld, and copied into target application as a stand-alone executable
- .a filename convention
- Benefits: No dependency problems (single executable file)
- Cost: Large file size, Library code possibly loaded multiple times in memory

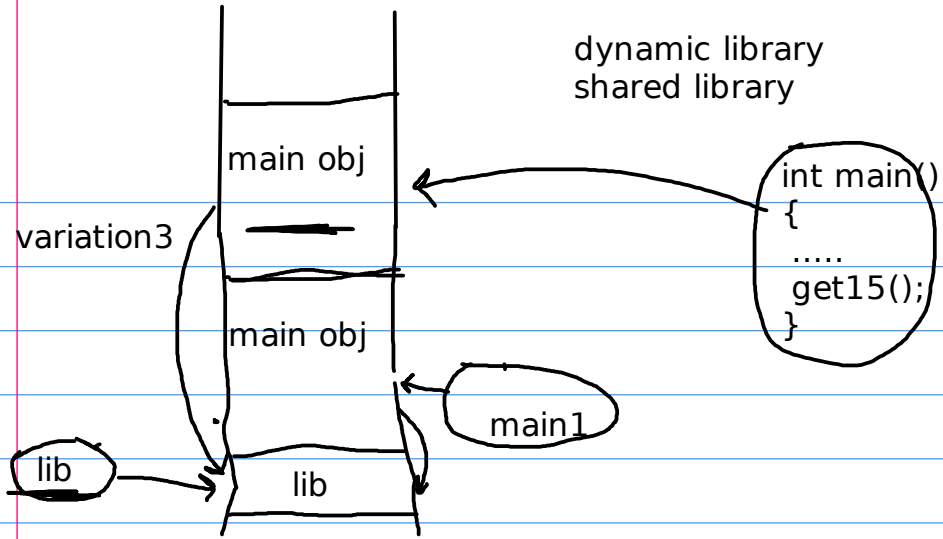


Linking Process

Static Linking

- Code and variables resolved at compile time by interpreter ld, and copied into target application as a stand-alone executable
- .a filename convention
- Benefits: No dependency problems (single executable file)
- Cost: Large file size, Library code possibly loaded multiple times in memory




dynamic library
shared library
main obj
variation3
main obj
int main()
{
.....
get15();
}
main1
lib
lib

Linking Process (cont.)

Dynamic Linking

- Code and variables resolved at load time by runtime interpreter ld-linux.so.2, and copied into target executable as symbols
- .so file convention
- Benefit: Small file size, library code loaded once in shared memory
- Cost: Dependency management