Steps for Introducing a System Call

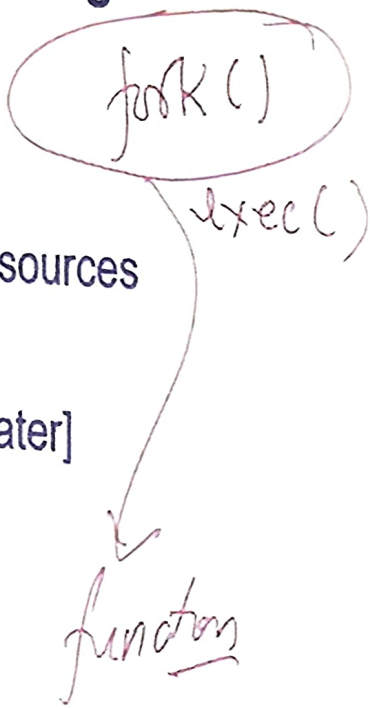
- Ensure that you are in the linux-xx.xx.y directory
- Create your own directory
  - **mkdir info**
    cd info/
  - **Create processInfo.h**
    - asmlinkage long sys_listProcessInfo(void);
  - Create processInfo.c
  - Write Makefile and change top level Makefile
    - obj-y:=listProcessInfo.o
    - core -y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ info/
  - Alter the /arch/x86/entry/syscalls/syscall_64.tbl
    - 548 common processInfo sys_processInfo
  - Alter /include/linux/syscalls.h
    - Add asmlinkage long sys_hello(void)
  - Compile
    - sudo make && sudo make modules_install && sudo make install
  - Test

Process

- Definition: **execution environment with restricted rights**
  - Address Space with One or More Threads
  - Owns memory (mapped pages)
  - Owns file descriptors, file system context, ...
  - Encapsulates one or more threads sharing process resources
- Application program executes as a process
  - Complex applications can fork/exec child processes [later]
- Why processes?
  - Protected from each other. OS Protected from them.
  - Execute concurrently [trade-offs with threads? later]
  - Basic unit OS deals with

Process

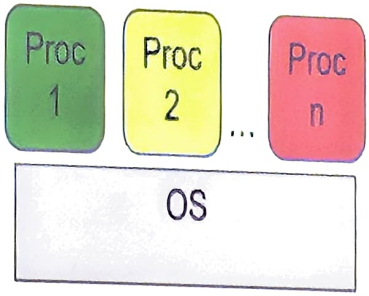
- Definition: **execution environment with restricted rights**
  - Address Space with One or More Threads
  - Owns memory (mapped pages)
  - Owns file descriptors, file system context, ...
  - Encapsulates one or more threads sharing process resources
- Application program executes as a process
  - Complex applications can fork/exec child processes [later]
- Why processes?
  - Protected from each other. OS Protected from them.
  - Execute concurrently [trade-offs with threads? later]
  - Basic unit OS deals with

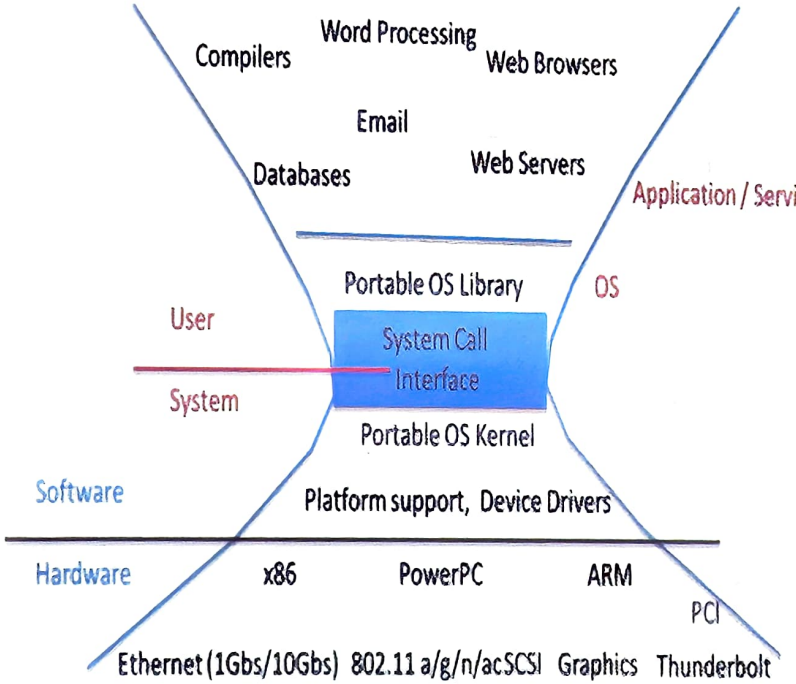


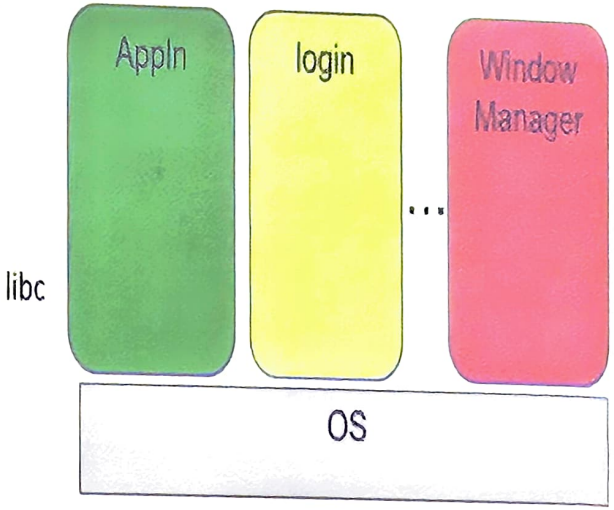
```
pid = fork()
if (pid == 0) do Something
    else
    exec(      )
```

OS Runtime Library

- The Runtime
  - Narrow Waist



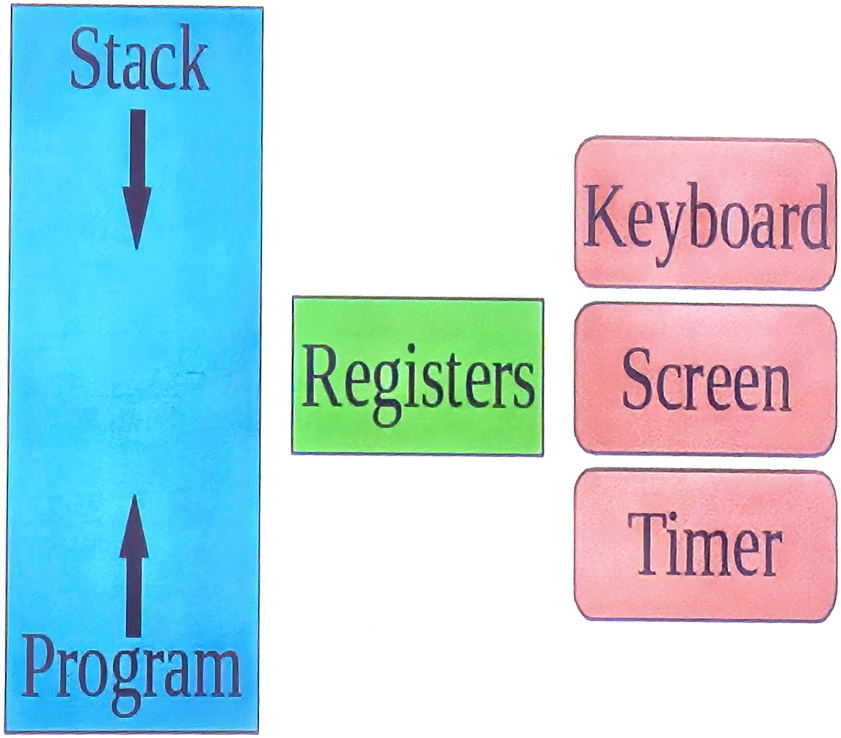




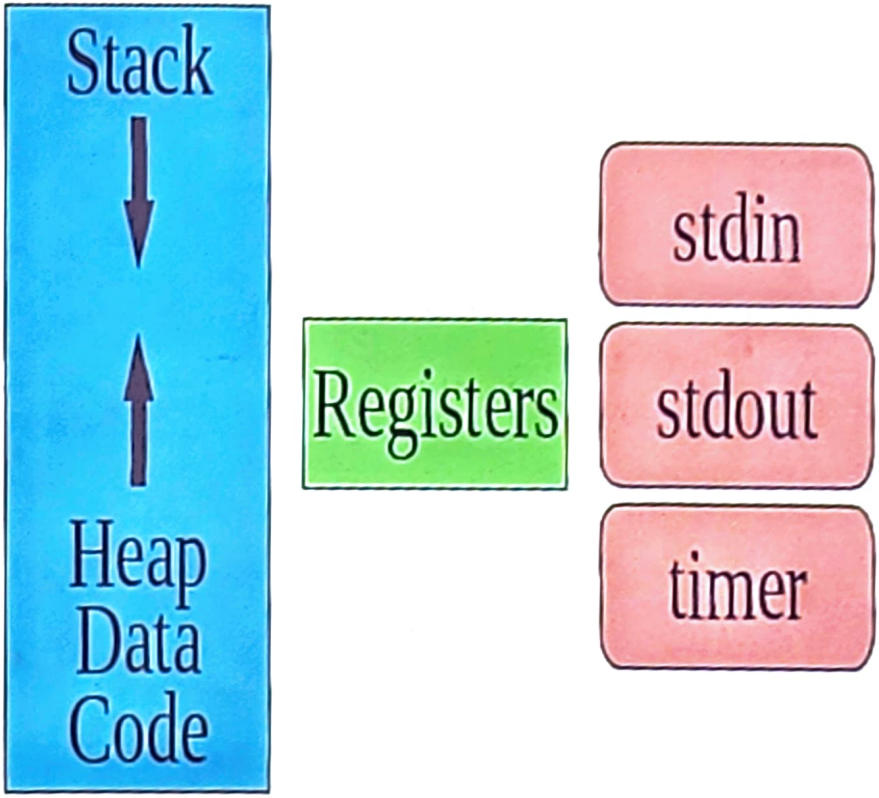
Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms *job* and *process* almost interchangeably
- Process – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section

The Bare Machine

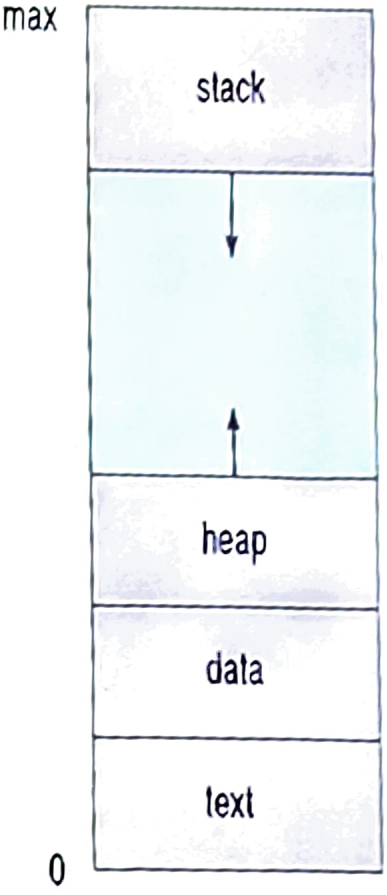


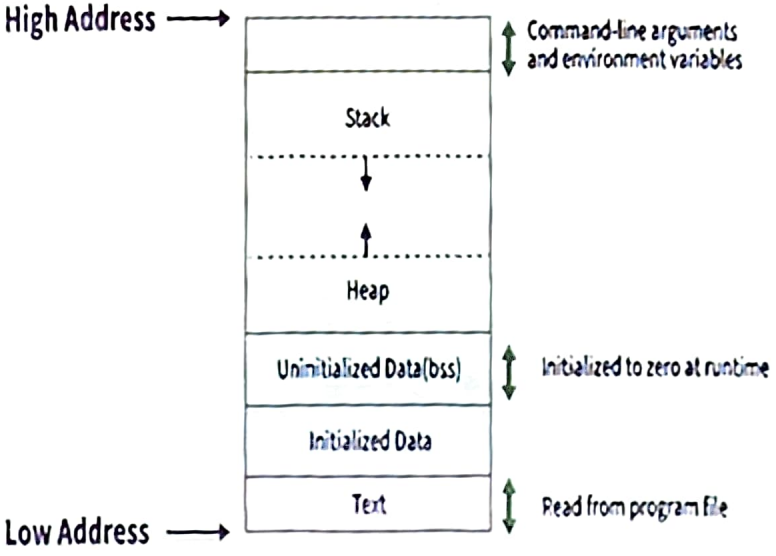
The Bare Machine



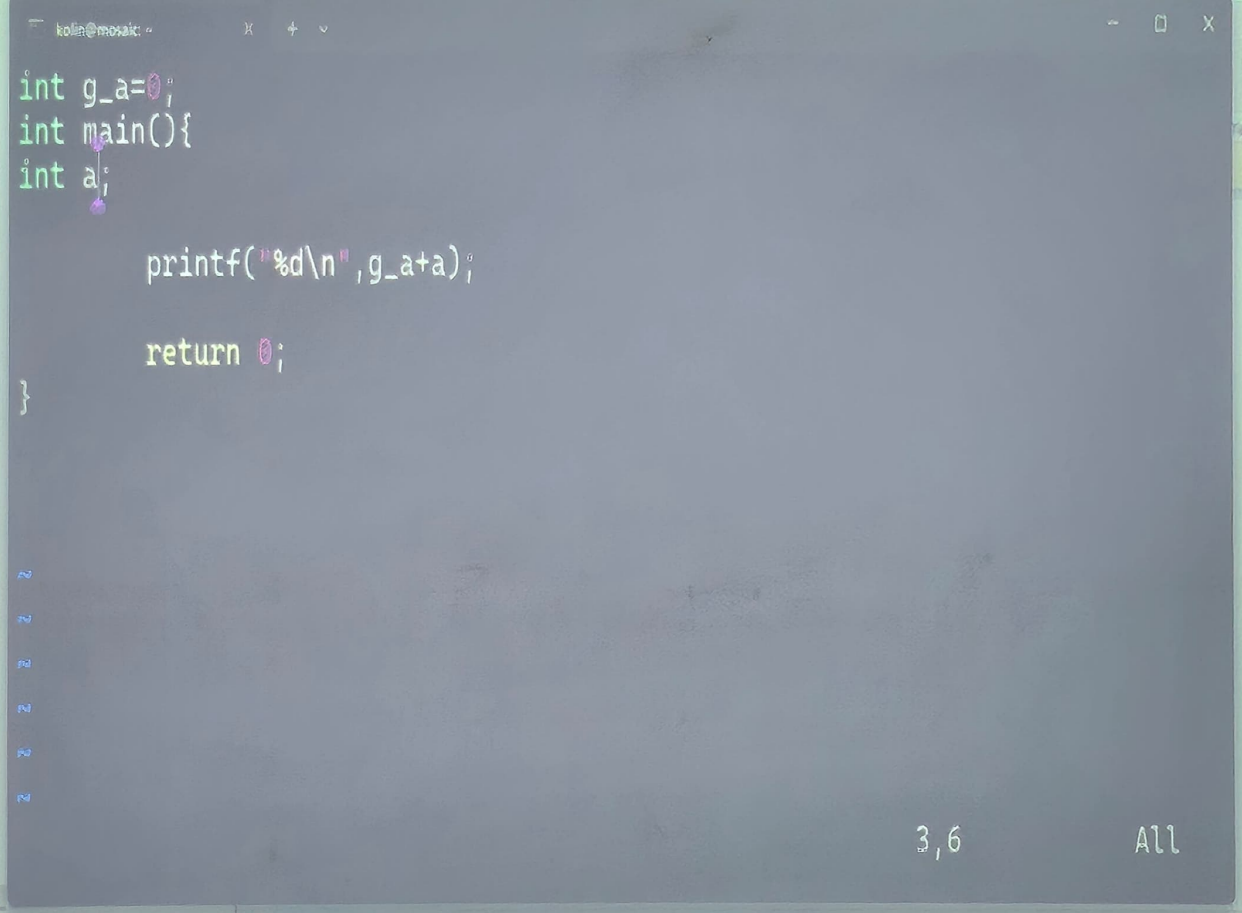
Process in Memory

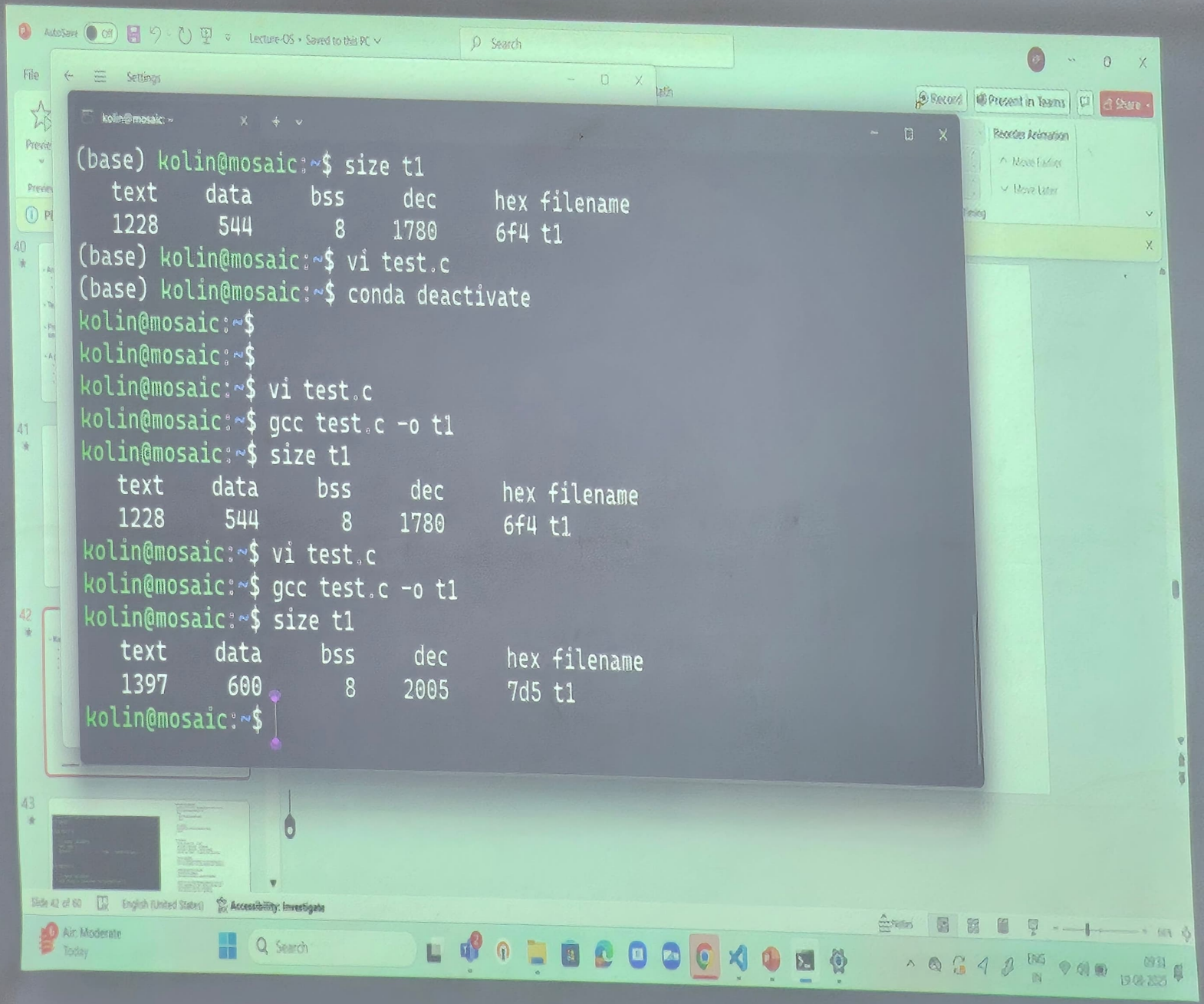
- **Kernel builds stack for new process**
  - Transfers argv[] and envp[] to top of new process
  - Hand-crafts stack frame for __main()
  - Sets registers
    - Stack pointer (to top frame)
    - Program counter (to start of __main())



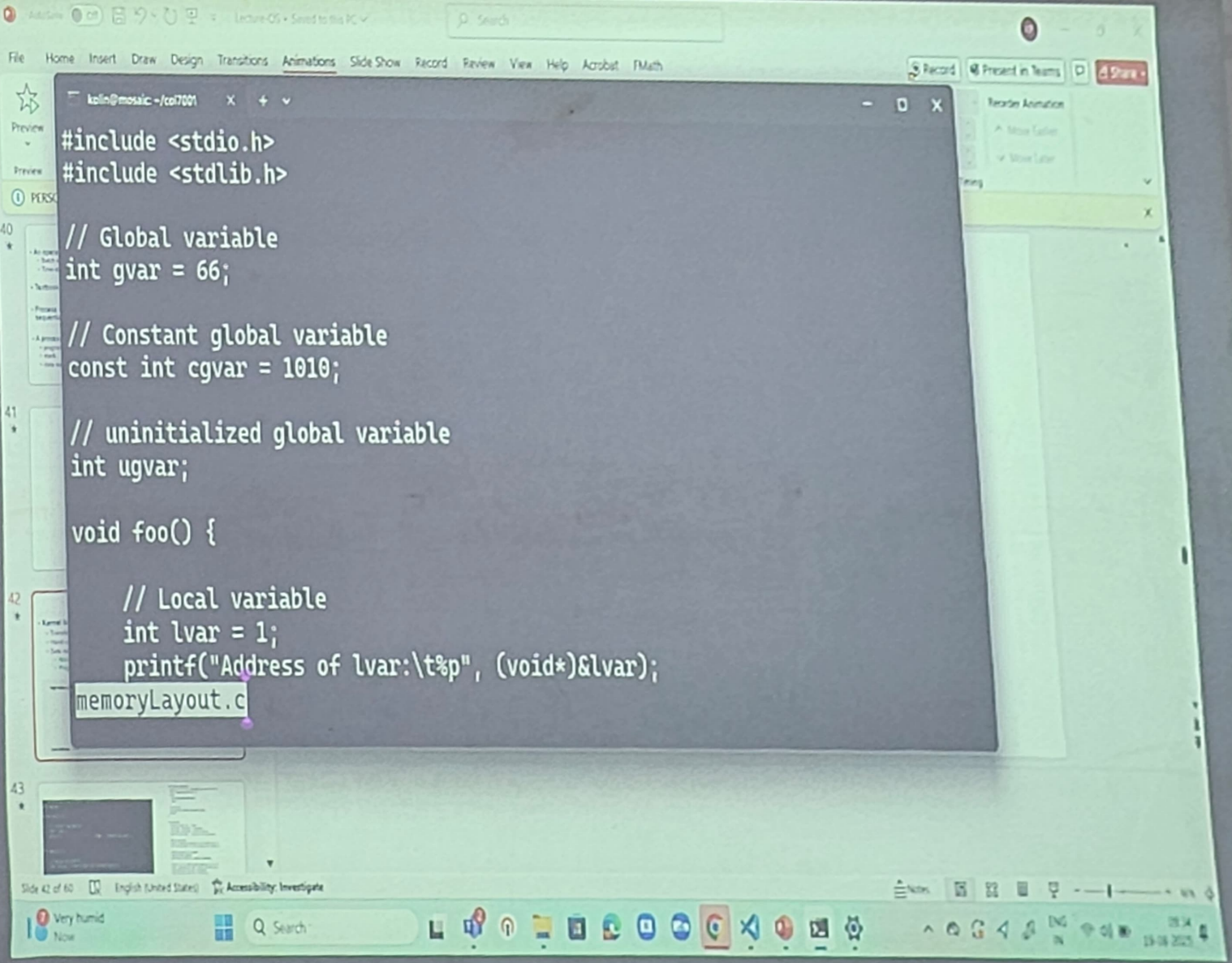


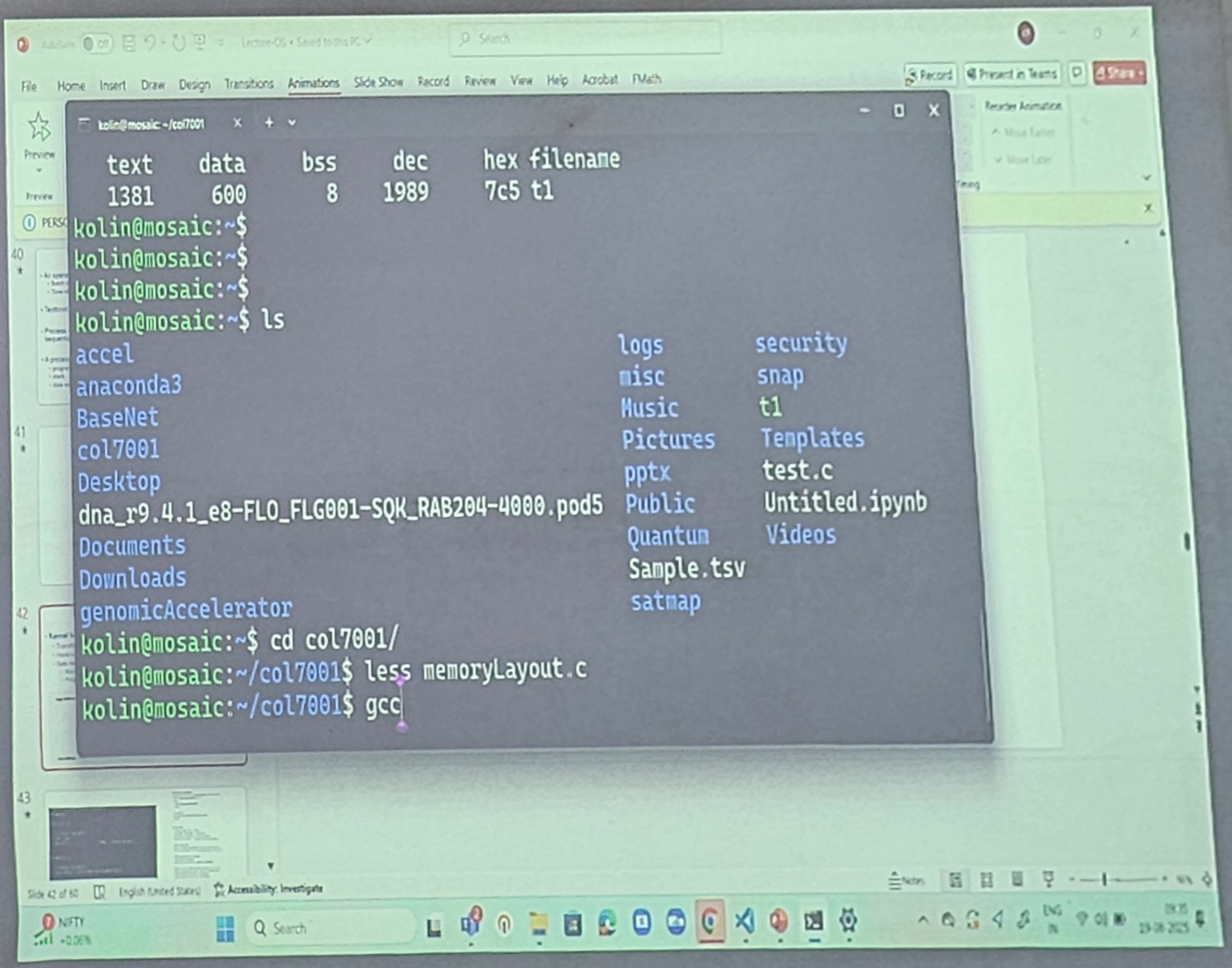
```
(base) kolin@mosaic:~$
(base) kolin@mosaic:~$ vi test.c
(base) kolin@mosaic:~$ gcc test.c -o t1
(base) kolin@mosaic:~$ size t1
   text    data     bss     dec     hex filename
   1228     544       8    1780     6f4 t1
(base) kolin@mosaic:~$ vi test.c
(base) kolin@mosaic:~$ gcc test.c -o t1
(base) kolin@mosaic:~$ size t1
   text    data     bss     dec     hex filename
   1228     544       8    1780     6f4 t1
(base) kolin@mosaic:~$ vi test.c

(base) kolin@mosaic:~$ gcc test.c -o t1
(base) kolin@mosaic:~$ size t1
   text    data     bss     dec     hex filename
   1228     544       8    1780     6f4 t1
(base) kolin@mosaic:~$
```

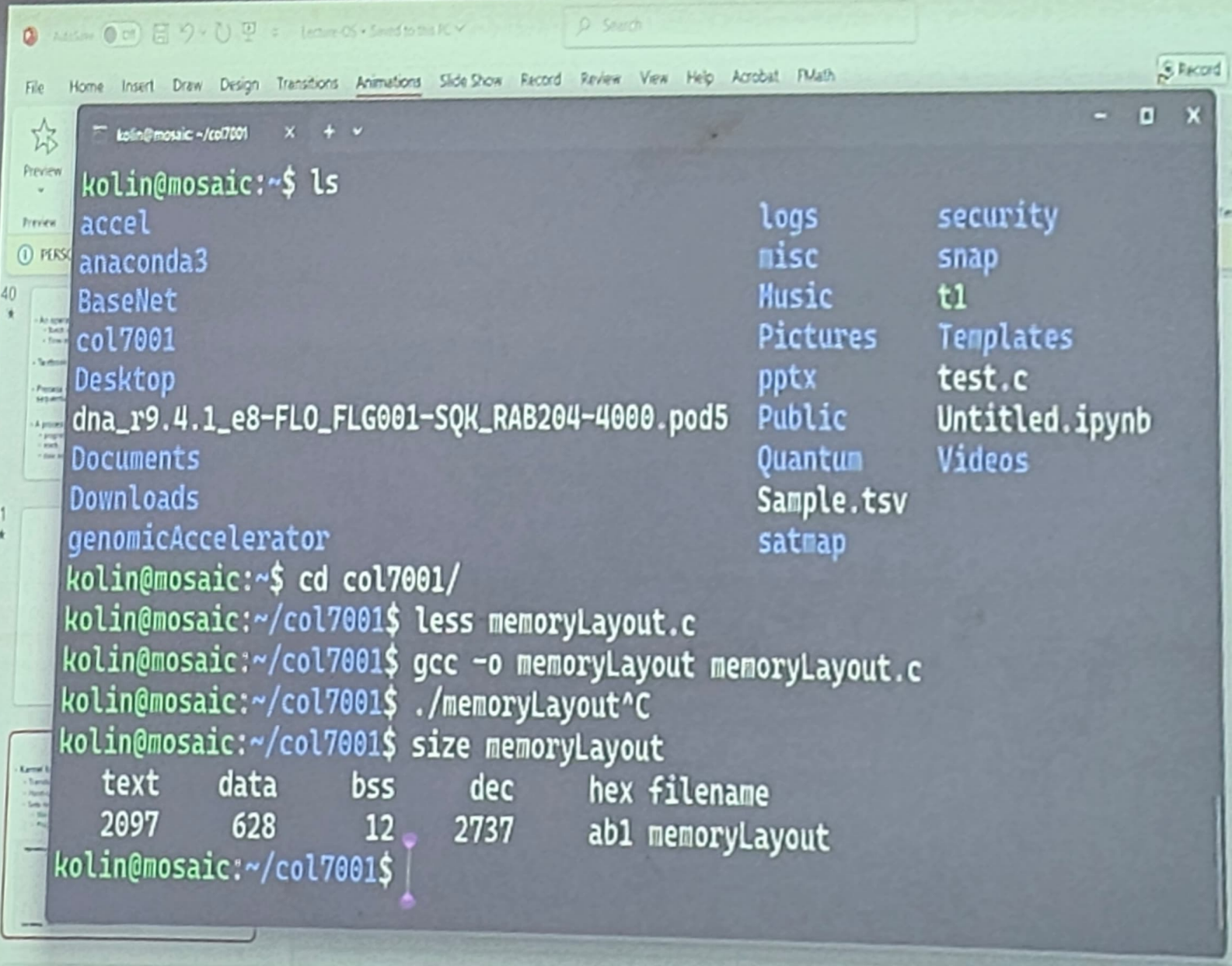

kolin@mosaic: ~
int g_a=0;
int main(){
int a;
printf("%d\n",g_a+a);
return 0;
}
3,6
All

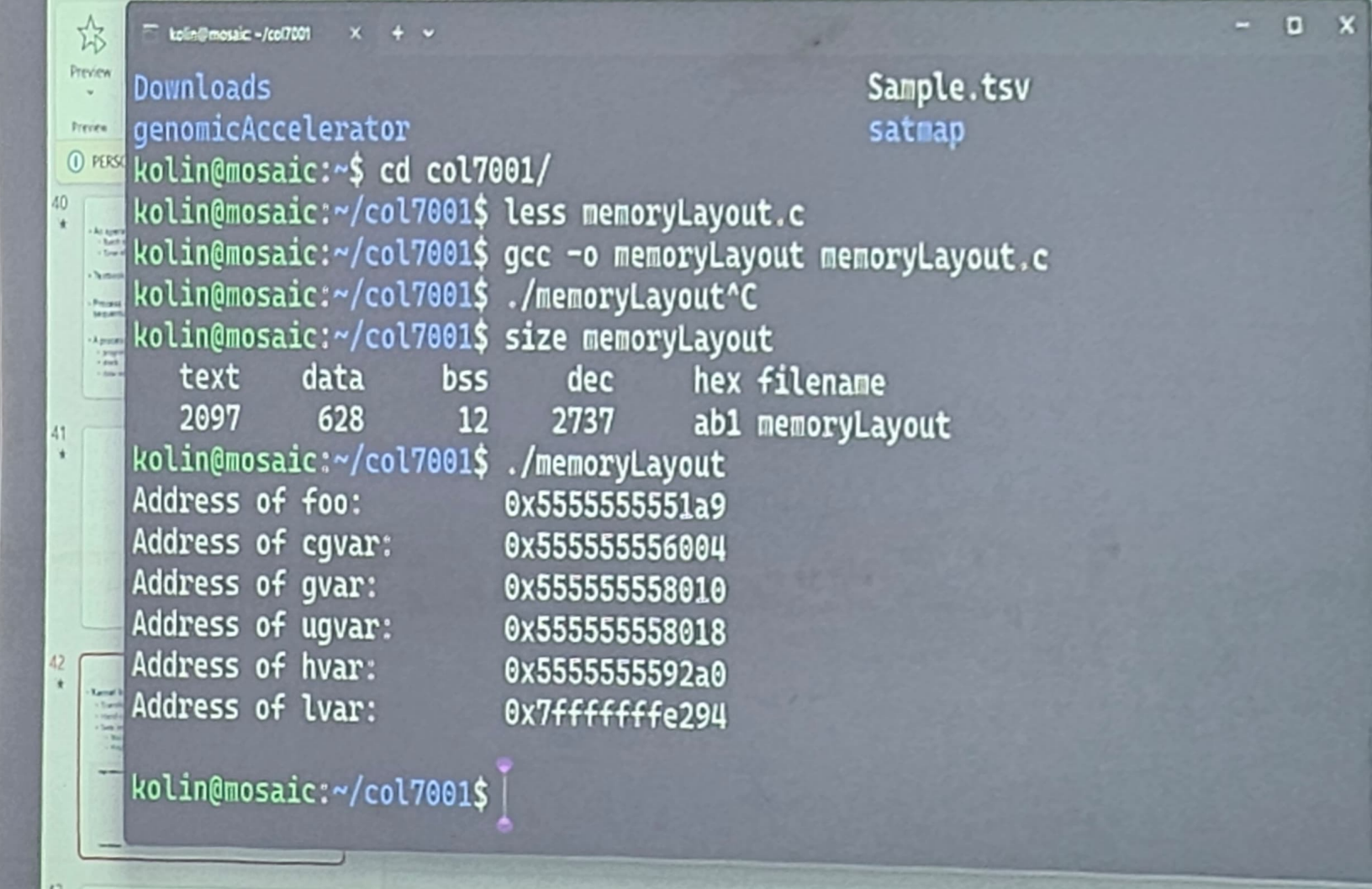

(base) kolin@mosaic:~$ size t1
text data bss dec hex filename
1228 544 8 1780 6f4 t1
(base) kolin@mosaic:~$ vi test.c
(base) kolin@mosaic:~$ conda deactivate
kolin@mosaic:~$
kolin@mosaic:~$
kolin@mosaic:~$ vi test.c
kolin@mosaic:~$ gcc test.c -o t1
kolin@mosaic:~$ size t1
text data bss dec hex filename
1228 544 8 1780 6f4 t1
kolin@mosaic:~$ vi test.c
kolin@mosaic:~$ gcc test.c -o t1
kolin@mosaic:~$ size t1
text data bss dec hex filename
1397 600 8 2005 7d5 t1
kolin@mosaic:~$

```c
// uninitialized global variable
int ugvar;

void foo() {

    // Local variable
    int lvar = 1;
    printf("Address of lvar:\t%p", (void*)&lvar);
}

int main() {

    // Heap variable
    int *hvar = (int*)malloc(sizeof(int));


    // Checking and comparing address of different
    // elements of program that should be stored in
    // different segements of the memory
    printf("Address of foo:\t\t%p\n", (void*)&foo);
    printf("Address of cgvar:\t%p\n", (void*)&cgvar);
    printf("Address of gvar:\t%p\n", (void*)&gvar);
    printf("Address of ugvar:\t%p\n", (void*)&ugvar);
    printf("Address of hvar:\t%p\n", (void*)hvar);
    foo();
    printf("\n\n");
        return 0;
}
```


kolin@mosaic ~/csi7001
#include <stdio.h>
#include <stdlib.h>
// Global variable
int gvar = 66;
// Constant global variable
const int cgvar = 1010;
// uninitialized global variable
int ugvar;
void foo() {
// Local variable
int lvar = 1;
printf("Address of lvar:\t%p", (void*)&lvar);
memoryLayout.c


kolin@mosaic: ~/col7001
text data bss dec hex filename
1381 600 8 1989 7c5 t1
kolin@mosaic:~$
kolin@mosaic:~$
kolin@mosaic:~$
kolin@mosaic:~$ ls
accel
anaconda3
BaseNet
col7001
Desktop
dna_r9.4.1_e8-FLO_FLG001-SQK_RAB204-4000.pod5
Documents
Downloads
genomicAccelerator
logs
misc
Music
Pictures
pptx
Public
Quantum
Sample.tsv
satmap
security
snap
t1
Templates
test.c
Untitled.ipynb
Videos
kolin@mosaic:~$ cd col7001/
kolin@mosaic:~/col7001$ less memoryLayout.c
kolin@mosaic:~/col7001$ gcc


File Home Insert Draw Design Transitions Animations Slide Show Record Review View Help Acrobat PMath
kolin@mosaic:~/col7001
kolin@mosaic:~$ ls
accel
anaconda3
BaseNet
col7001
Desktop
dna_r9.4.1_e8-FLO_FLG001-SQK_RAB204-4000.pod5
Documents
Downloads
genomicAccelerator
logs
misc
Music
Pictures
pptx
Public
Quantum
Sample.tsv
satmap
security
snap
t1
Templates
test.c
Untitled.ipynb
Videos
kolin@mosaic:~$ cd col7001/
kolin@mosaic:~/col7001$ less memoryLayout.c
kolin@mosaic:~/col7001$ gcc -o memoryLayout memoryLayout.c
kolin@mosaic:~/col7001$ ./memoryLayout^C
kolin@mosaic:~/col7001$ size memoryLayout
text data bss dec hex filename
2097 628 12 2737 ab1 memoryLayout
kolin@mosaic:~/col7001$


kolin@mosaic: ~/col7001
Downloads
genomicAccelerator
Sample.tsv
satmap
kolin@mosaic:~$ cd col7001/
kolin@mosaic:~/col7001$ less memoryLayout.c
kolin@mosaic:~/col7001$ gcc -o memoryLayout memoryLayout.c
kolin@mosaic:~/col7001$ ./memoryLayout^C
kolin@mosaic:~/col7001$ size memoryLayout
text data bss dec hex filename
2097 628 12 2737 ab1 memoryLayout
kolin@mosaic:~/col7001$ ./memoryLayout
Address of foo: 0x5555555551a9
Address of cgvar: 0x555555556004
Address of gvar: 0x555555558010
Address of ugvar: 0x555555558018
Address of hvar: 0x5555555592a0
Address of lvar: 0x7fffffffe294
kolin@mosaic:~/col7001$

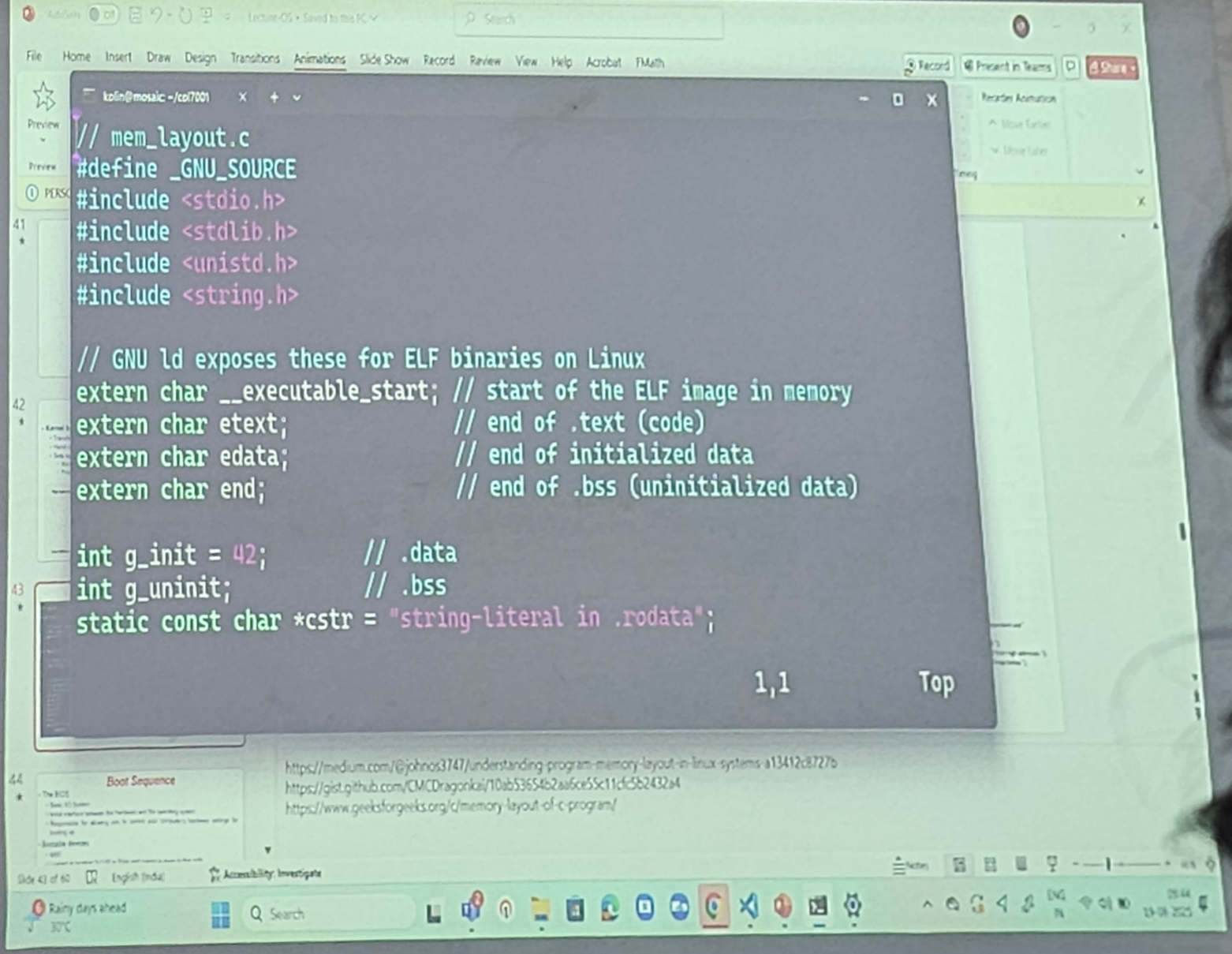
```c
// uninitialized global variable
int ugvar;

void foo() {

    // Local variable
    int lvar = 1;
    printf("Address of lvar:\t%p", (void*)&lvar);
}

int main() {

    // Heap variable
    int *hvar = (int*)malloc(sizeof(int));


    // Checking and comparing address of different
    // elements of program that should be stored in
    // different segements of the memory
    printf("Address of foo:\t\t%p\n", (void*)&foo);
    printf("Address of cgvar:\t%p\n", (void*)&cgvar);
    printf("Address of gvar:\t%p\n", (void*)&gvar);
    printf("Address of ugvar:\t%p\n", (void*)&ugvar);
    printf("Address of hvar:\t%p\n", (void*)hvar);
    foo();
    printf("\n\n");
    return 0;
}
```

```c
static void print_maps(void) {
  puts("\n================ /proc/self/maps ================");
  FILE *f = fopen("/proc/self/maps", "r");
  if (!f) {
    perror("fopen(/proc/self/maps)");
    return;
  }
  char line[512];
  while (fgets(line, sizeof line, f)) fputs(line, stdout);
  fclose(f);
}

int main(void) {
  int local_on_stack = 123;     // stack
  void *heap1 = malloc(1024);   // heap probe
  void *heap2 = malloc(1024);   // another allocation
  void *brk_now = sbrk(0);      // program break (heap end hint)

  char exe_path[4096];
  ssize_t n = readlink("/proc/self/exe", exe_path, sizeof exe_path - 1);
  if (n >= 0) { exe_path[n] = '\0'; } else { strcpy(exe_path, "unknown"); }

  printf("Executable: %s\n", exe_path);
  printf("PID: %d\n", getpid());
  printf("Page size: %ld bytes\n", sysconf(_SC_PAGESIZE));

  puts("\n========== High-level regions (addresses) ==========");
  printf("__executable_start: %p\n", (void*)&__executable_start);
  printf("etext (end of .text): %p\n", (void*)&etext);
  printf("edata (end of .data): %p\n", (void*)&edata);
  printf("end   (end of .bss):  %p\n", (void*)&end);

  puts("\n---------- Probes (examples in those regions) ----------");
  printf(".text  main     : %p\n", (void*)&main);
  printf(".text  foo      : %p\n", (void*)&foo);
  printf(".rodata cstr    : %p -> \"%s\"\n", (void*)cstr, cstr);
  printf(".data  g_init   : %p (=%d)\n", (void*)&g_init, g_init);
  printf(".bss   g_uninit : %p\n", (void*)&g_uninit);
  printf("heap   malloc(1k): %p, %p\n", heap1, heap2);
  printf("heap   program brk: %p (sbrk(0))\n", brk_now);
  printf("stack  local var : %p (&local_on_stack)\n", (void*)&local_on_stack);

  puts("\n---------- Layout notes ----------");
  puts("* On most x86_64 Linux builds with PIE, the executable is position-independent and *"
       "  the loader maps segments at randomized base addresses (ASLR).");
  puts("* __executable_start <= .text <= etext <= .data <= edata <= .bss <= end.");
  puts("* Heap grows upward from the program break; stack grows downward from high addresses.");
  puts("* For the authoritative mapping (including shared libs), see /proc/self/maps below.");

  print_maps();

  free(heap2);
  free(heap1);
  return 0;
}
```

```
static void print_maps(void) {
  puts("\n================ /proc/self/maps ================");
  FILE *f = fopen("/proc/self/maps", "r");
  if (!f) {
    perror("fopen(/proc/self/maps)");
    return;
  }
  char line[512];
  while (fgets(line, sizeof line, f)) fputs(line, stdout);
  fclose(f);
}

int main(void) {
  int local_on_stack = 123;     // stack
  void *heap1 = malloc(1024);   // heap probe
  void *heap2 = malloc(1024);   // another allocation
  void *brk_now = sbrk(0);      // program break (heap end hint)

  char exe_path[4096];
  ssize_t n = readlink("/proc/self/exe", exe_path, sizeof exe_path - 1);
  if (n >= 0) { exe_path[n] = '\0'; } else { strcpy(exe_path, "unknown"); }

  printf("Executable: %s\n", exe_path);
  printf("PID: %d\n", getpid());
  printf("Page size: %ld bytes\n", sysconf(_SC_PAGESIZE));
```

```
int local_on_stack = 123;     // stack
void *heap1 = malloc(1024);   // heap probe
void *heap2 = malloc(1024);   // another allocation
void *brk_now = sbrk(0);      // program break (heap end hint)

char exe_path[4096];
ssize_t n = readlink("/proc/self/exe", exe_path, sizeof exe_path - 1);
if (n >= 0) { exe_path[n] = '\0'; } else { strcpy(exe_path, "unknown"); }

printf("Executable: %s\n", exe_path);
printf("PID: %d\n", getpid());
printf("Page size: %ld bytes\n", sysconf(_SC_PAGESIZE));

puts("\n========== High-level regions (addresses) ==========");
printf("__executable_start : %p\n", (void*)&__executable_start);
printf("etext (end of .text): %p\n", (void*)&etext);
printf("edata (end of .data): %p\n", (void*)&edata);
printf("end   (end of .bss) : %p\n", (void*)&end);
```

```
puts("\n========== High-level regions (addresses) ==========");
printf("__executable_start : %p\n", (void*)&__executable_start);
printf("etext (end of .text): %p\n", (void*)&etext);
printf("edata (end of .data): %p\n", (void*)&edata);
printf("end   (end of .bss) : %p\n", (void*)&end);

puts("\n---------- Probes (examples in those regions) ----------");
printf(".text   main     : %p\n", (void*)&main);
printf(".text   foo      : %p\n", (void*)&foo);
printf(".rodata cstr     : %p -> \"%s\"\n", (void*)cstr, cstr);
printf(".data   g_init   : %p (=%d)\n", (void*)&g_init, g_init);
printf(".bss    g_uninit : %p\n", (void*)&g_uninit);
printf("heap    malloc(1k) : %p, %p\n", heap1, heap2);
printf("heap    program brk: %p (sbrk(0))\n", brk_now);
printf("stack   local var  : %p (&local_on_stack)\n", (void*)&local_on_stack);

puts("\n---------- Layout notes ----------");
puts("* On most x86_64 Linux builds with PIE, the executable is position-independent and
   "the loader maps segments at randomized base addresses (ASLR).");
puts("* __executable_start <= .text <= etext <= .data <= edata <= .bss <= end.");
puts("* Heap grows upward from the program break; stack grows downward from high ad
```


kolin@mosaic ~/cpl7001
// mem_layout.c
#define _GNU_SOURCE
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
// GNU ld exposes these for ELF binaries on Linux
extern char __executable_start; // start of the ELF image in memory
extern char etext; // end of .text (code)
extern char edata; // end of initialized data
extern char end; // end of .bss (uninitialized data)
int g_init = 42; // .data
int g_uninit; // .bss
static const char *cstr = "string-literal in .rodata";
1,1 Top
Boot Sequence
https://medium.com/@johnos3747/understanding-program-memory-layout-in-linux-systems-a13412c8727b
https://gist.github.com/CMCDragonkai/10ab53654b2aa6ce55c11cfc5b2432a4
https://www.geeksforgeeks.org/c/memory-layout-of-c-program/

```c
static const char *cstr = "string-literal in .rodata";

static int foo(int x) { // lives in .text
    return x + 1;
}


static void print_maps(void) {
    puts("\n=============== /proc/self/maps ===============");
    FILE *f = fopen("/proc/self/maps", "r");
    if (!f) {
        perror("fopen(/proc/self/maps)");
        return;
    }
    char line[512];
    while (fgets(line, sizeof line, f)) fputs(line, stdout);
    fclose(f);
}
```

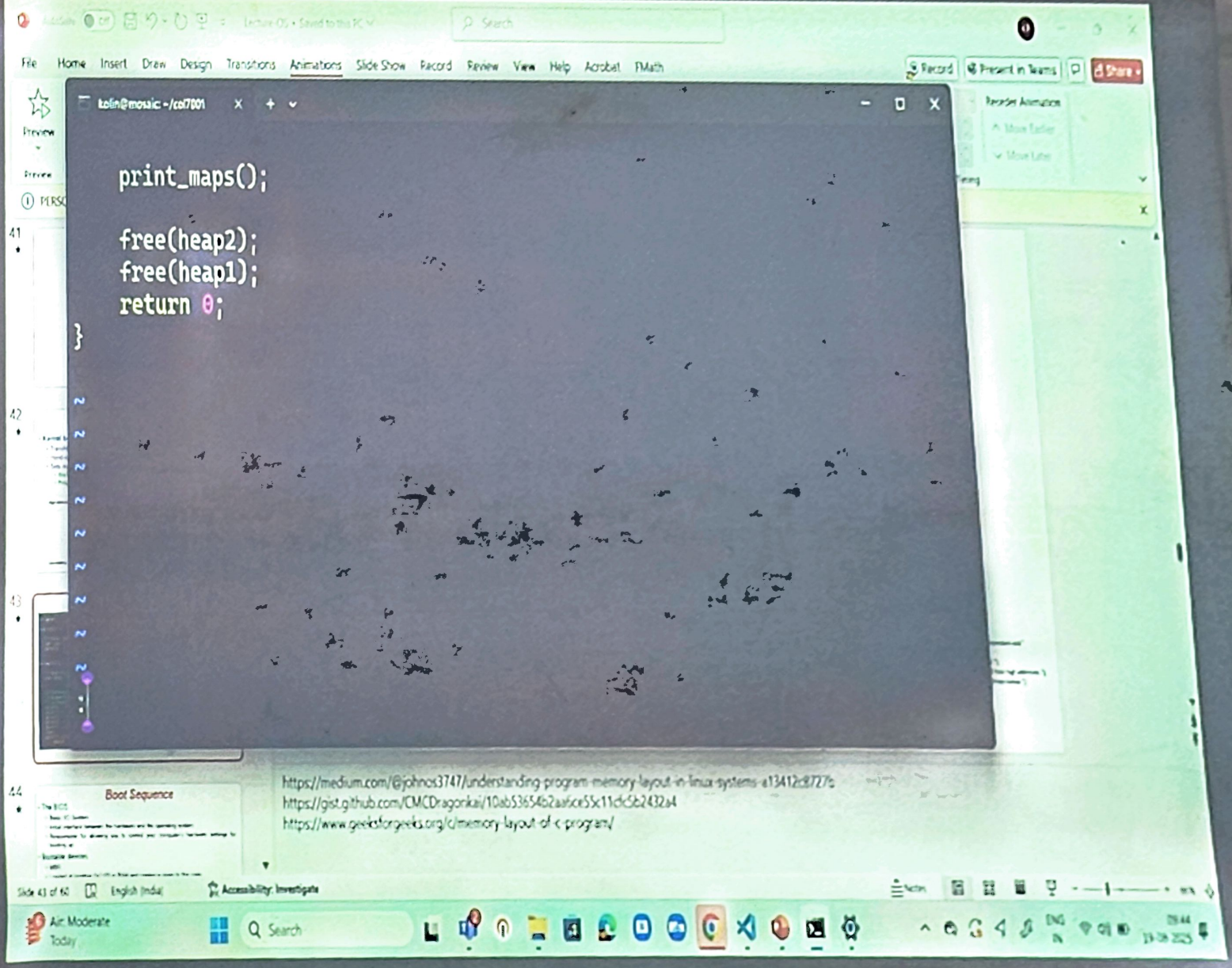
21,0-1 25%

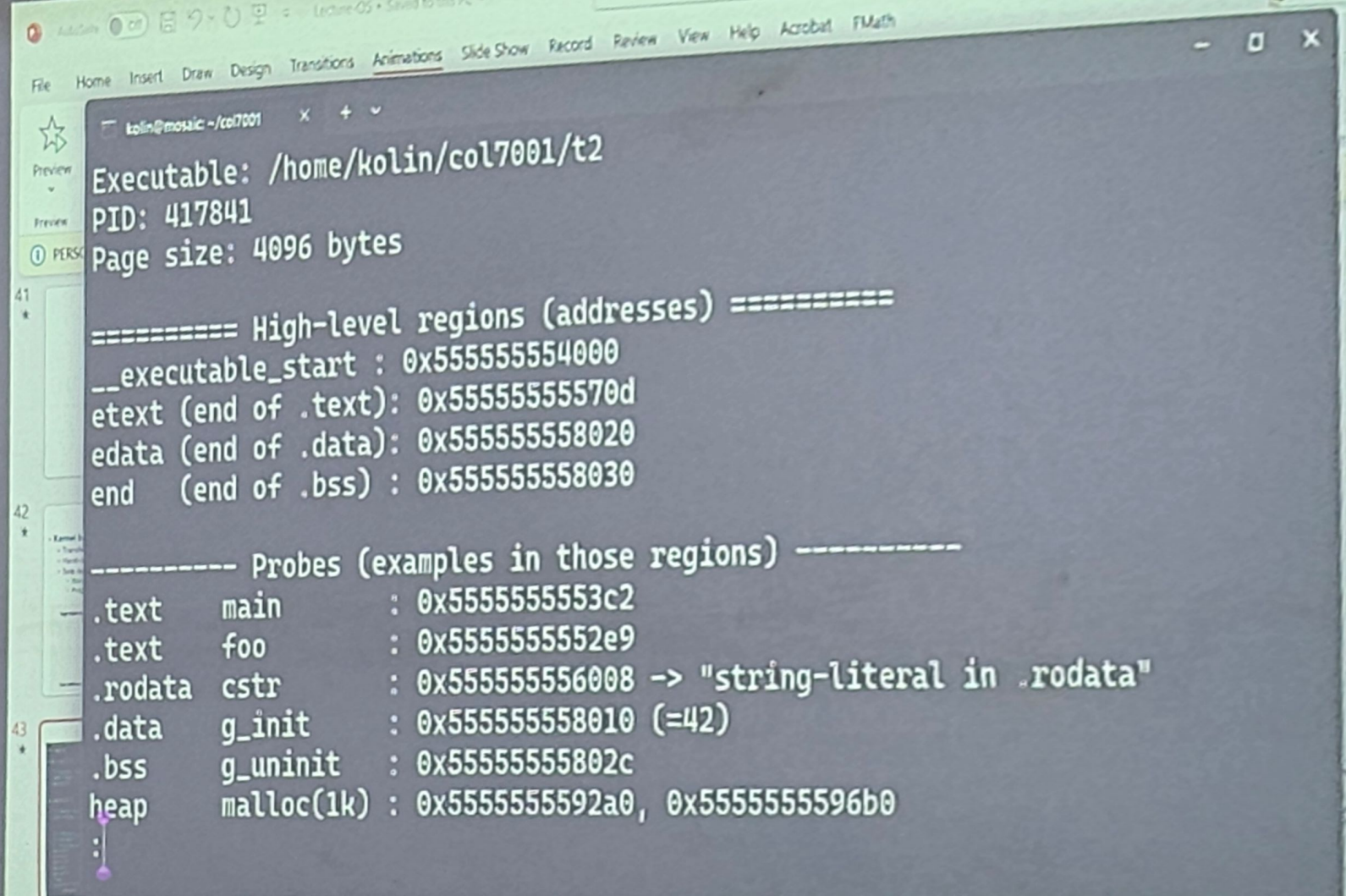
https://medium.com/@johnos3747/understanding-program-memory-layout-in-linux-systems-a13412c8727b
https://gist.github.com/CMCDragonkai/10ab53654b2aa6ce55c11cfc5b2432a4
https://www.geeksforgeeks.org/c/memory-layout-of-c-program/

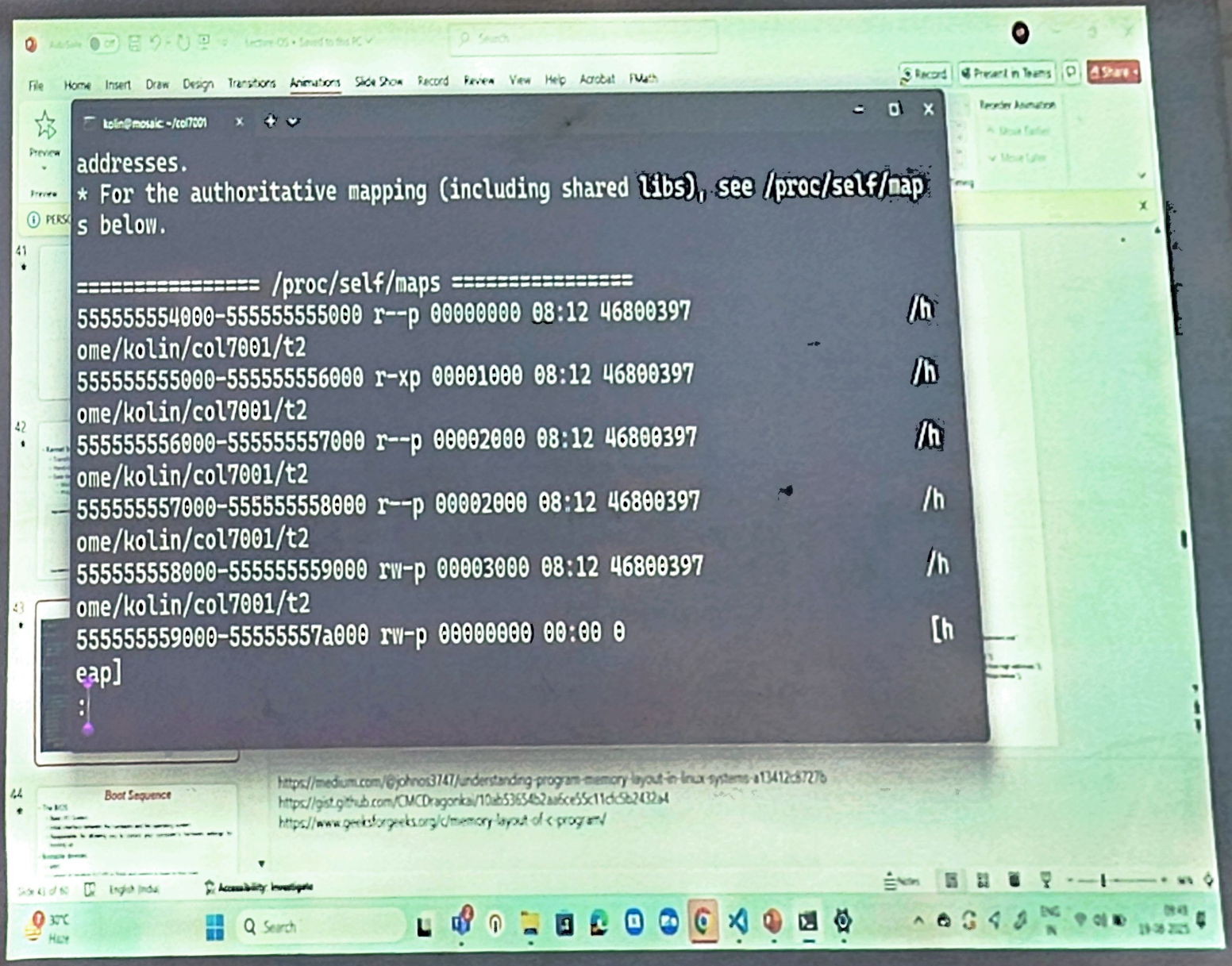
kolin@mosaic: ~/col7001

```
    printf("PID: %d\n", getpid());
    printf("Page size: %ld bytes\n", sysconf(_SC_PAGESIZE));

    puts("\n========== High-level regions (addresses) ==========");
    printf("__executable_start : %p\n", (void*)&__executable_start);
    printf("etext (end of .text): %p\n", (void*)&etext);
    printf("edata (end of .data): %p\n", (void*)&edata);
    printf("end   (end of .bss) : %p\n", (void*)&end);

    puts("\n---------- Probes (examples in those regions) ----------");
    printf(".text    main       : %p\n", (void*)&main);
    printf(".text    foo        : %p\n", (void*)&foo);
    printf(".rodata  cstr       : %p -> \"%s\"\n", (void*)cstr, cstr);
    printf(".data    g_init     : %p (=%d)\n", (void*)&g_init, g_init);
    printf(".bss     g_uninit   : %p\n", (void*)&g_uninit);
    printf("heap     malloc(1k) : %p, %p\n", heap1, heap2);
    printf("heap     program brk: %p (sbrk(0))\n", brk_now);
```

50,5 73%


kolin@mosaic: ~/cs7001
print_maps();
free(heap2);
free(heap1);
return 0;
}
https://medium.com/@johncs3747/understanding-program-memory-layout-in-linux-systems-a13412c8727c
https://gist.github.com/CMCDragonkai/10ab53654b2aa6ce55c11cfc5b2432a4
https://www.geeksforgeeks.org/c/memory-layout-of-c-program/
Boot Sequence


File Home Insert Draw Design Transitions Animations Slide Show Record Review View Help Acrobat FMath
kolin@mosaic: ~/col7001
Executable: /home/kolin/col7001/t2
PID: 417841
Page size: 4096 bytes
========== High-level regions (addresses) ==========
__executable_start : 0x555555554000
etext (end of .text): 0x55555555570d
edata (end of .data): 0x555555558020
end (end of .bss) : 0x555555558030
---------- Probes (examples in those regions) ----------
.text main : 0x5555555553c2
.text foo : 0x5555555552e9
.rodata cstr : 0x555555556008 -> "string-literal in .rodata"
.data g_init : 0x555555558010 (=42)
.bss g_uninit : 0x55555555802c
heap malloc(1k) : 0x5555555592a0, 0x5555555596b0


kolin@mosaic: ~/col7001
addresses.
* For the authoritative mapping (including shared libs), see /proc/self/nap
s below.
========== /proc/self/maps ==========
555555554000-555555555000 r--p 00000000 08:12 46800397 /h
ome/kolin/col7001/t2
555555555000-555555556000 r-xp 00001000 08:12 46800397 /h
ome/kolin/col7001/t2
555555556000-555555557000 r--p 00002000 08:12 46800397 /h
ome/kolin/col7001/t2
555555557000-555555558000 r--p 00002000 08:12 46800397 /h
ome/kolin/col7001/t2
555555558000-555555559000 rw-p 00003000 08:12 46800397 /h
ome/kolin/col7001/t2
555555559000-55555557a000 rw-p 00000000 00:00 0 [h
eap]
:
Boot Sequence
https://medium.com/@johnosi3747/understanding-program-memory-layout-in-linux-systems-a13412c8727b
https://gist.github.com/CMCDragonkai/10ab53654b2aa6ce55c11cfc5b2432a4
https://www.geeksforgeeks.org/c/memory-layout-of-c-program/