# Computer System Organization
# [Spring 18, Mu]

R3: more pointers, and seg faults

# Clab grades

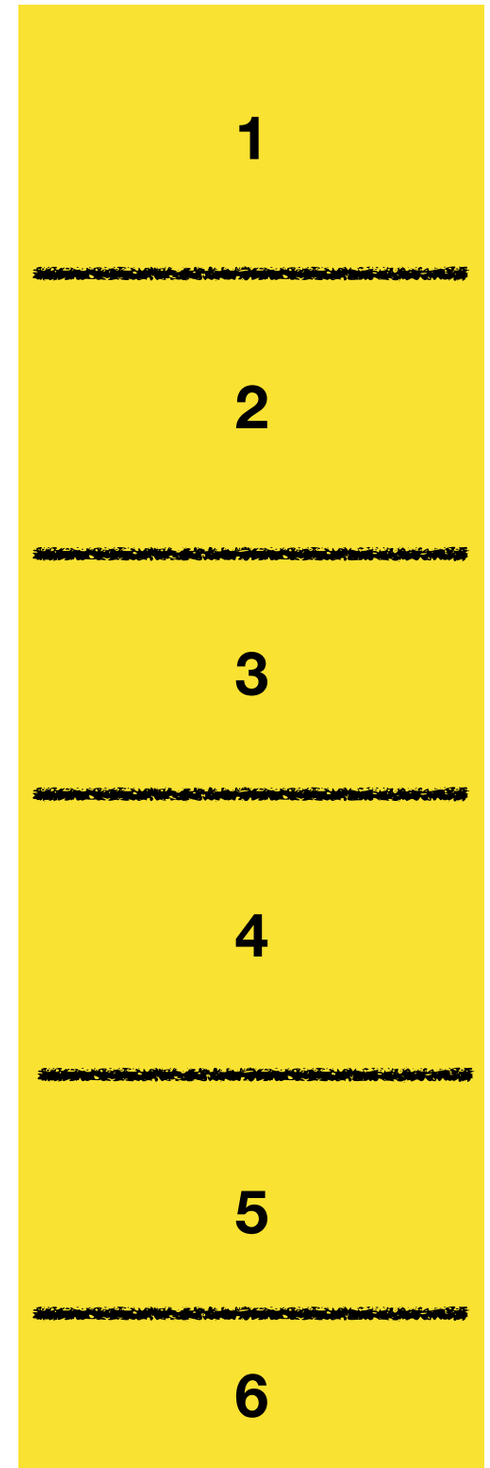
- Lab parts will be graded together
  - Last part was due last night at midnight
    - With 5 possible grace days, earliest possible grading time is this Sunday
    - Feel free to ask about part1 in office hours now though

# Array access and pointer

```
int main (int argc, char ** argv) {
    int my_array[] = {1, 2, 3, 4, 5, 6};
    printf("%d", *(my_array + 4));
    printf("%d", my_array[4]);
    printf("%d", 4[my_array]);
}
```

*(my_array + 4)

*(4 + my_array)




0x1000F000 (my_array)

0x1000F004

0x1000F008

0x1000F00C

0x1000F010

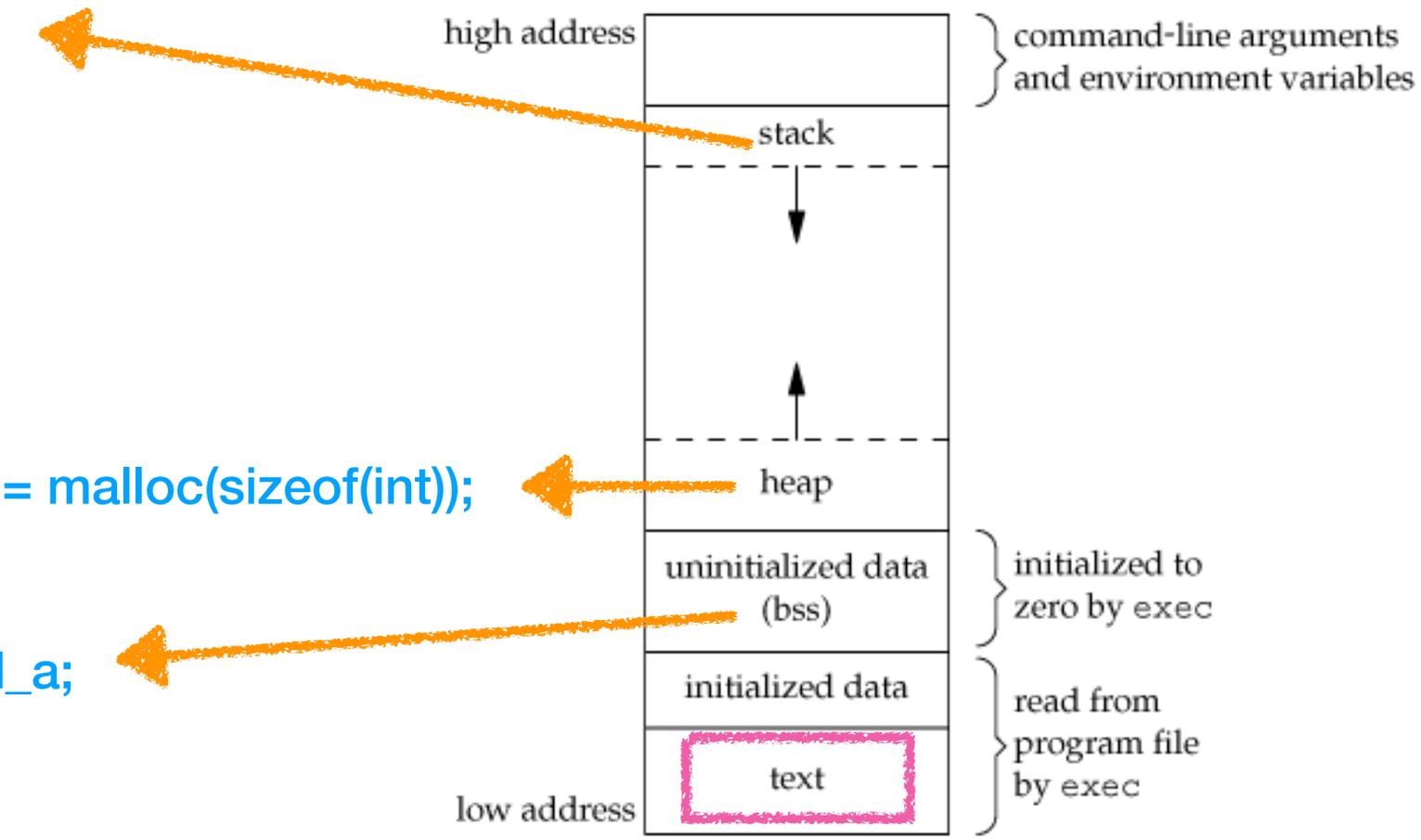
0x1000F014

# Memory layout of C programs

```
{
  int a;
}
```

```
{
  int* ptr = malloc(sizeof(int));
}
```

```
int global_a;
```

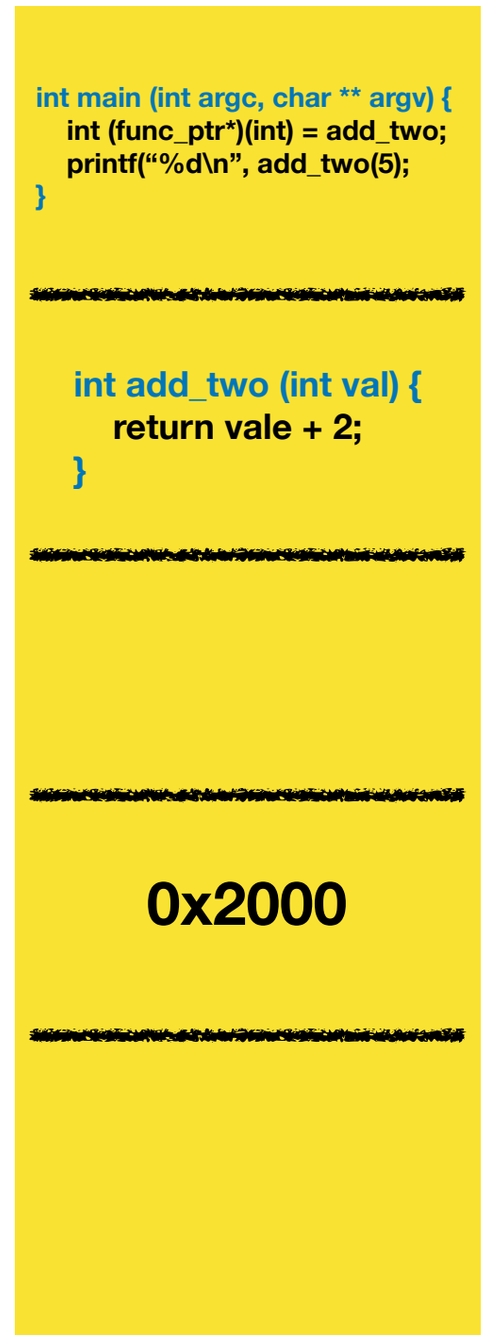


# Function Pointer in C

- A function pointer is just a variable with a unique interpretation.
  - The memory address(ADDR) of that variable.
  - The number of bytes used by the variable.
  - How to interpret the content stored in ADDR.
    - It represents a memory address.
    - The content in the memory address is part of the program.

# Function Pointer in C

```
int add_two (int val) {
    return vale + 2;
}

int main (int argc, char ** argv) {
    int (*func_ptr)(int) = add_two;
    printf("%d\n", func_ptr(5);
}
```




0x1000
(main)

0x2000
(add_two)

0x1000F008
(func_ptr)

0x1000F00C

# Function object in C and Python

```c
int add_two (int val) {
    return val + 2;
}

int main (int argc, char ** argv) {
    int (*func_ptr)(int) = add_two;
    printf("%d\n", add_two(5);
}
```

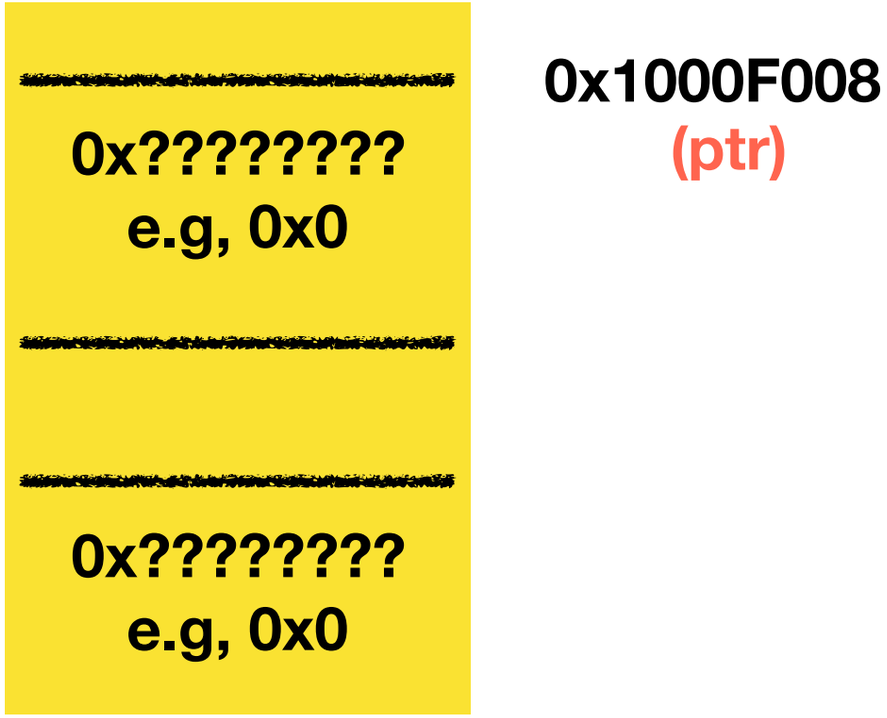
```python
def add_two(val):
    return val + 2

def main():
    func_ptr = add_two
    print(func_ptr(5))

if __name__ == '__main__':
    main()
```

# Segmentation Fault

- In computing, a segmentation fault(often shortened to segfault) or access violation is a fault, or failure condition, raised by hardware with memory protection, notifying an operation system the software has attempted to access a restricted area of memory(a memory access violation) —— Wikipedia

```
int main (int argc, char ** argv) {
    int val = 5;
    int *ptr;
    printf("%d\n", *ptr);
}
```

```
int *ptr = malloc(sizeof(int));
```



# C v.s Python (variable)

```c
int main (int argc, char **argv) {
    void* var = null;

    char* __temp1 = “12345”;
    var = (void*)__temp1;
    printf(“%s\n”, (char*) var);

    float __temp2 = 1.5;
    var = (void*) &__temp2;
    printf(“%f\n”, *((float*) var));

}
```

```python
def main():
    var = “12345”
    print(var)

    var = 1.5
    print(var)
```

# Object like structs with function pointers

```
struct obj {
   int count;
   Int (*get_count)(struct obj self);
}


struct obj my_o;
/* some initialization */

printf("Count %d\n",
     my_o->get_count(my_o));
```

```
class obj:
   count = 4
   def get_count(self):
        return self.count


my_o = obj();

print("Count", my_o.get_count())
```