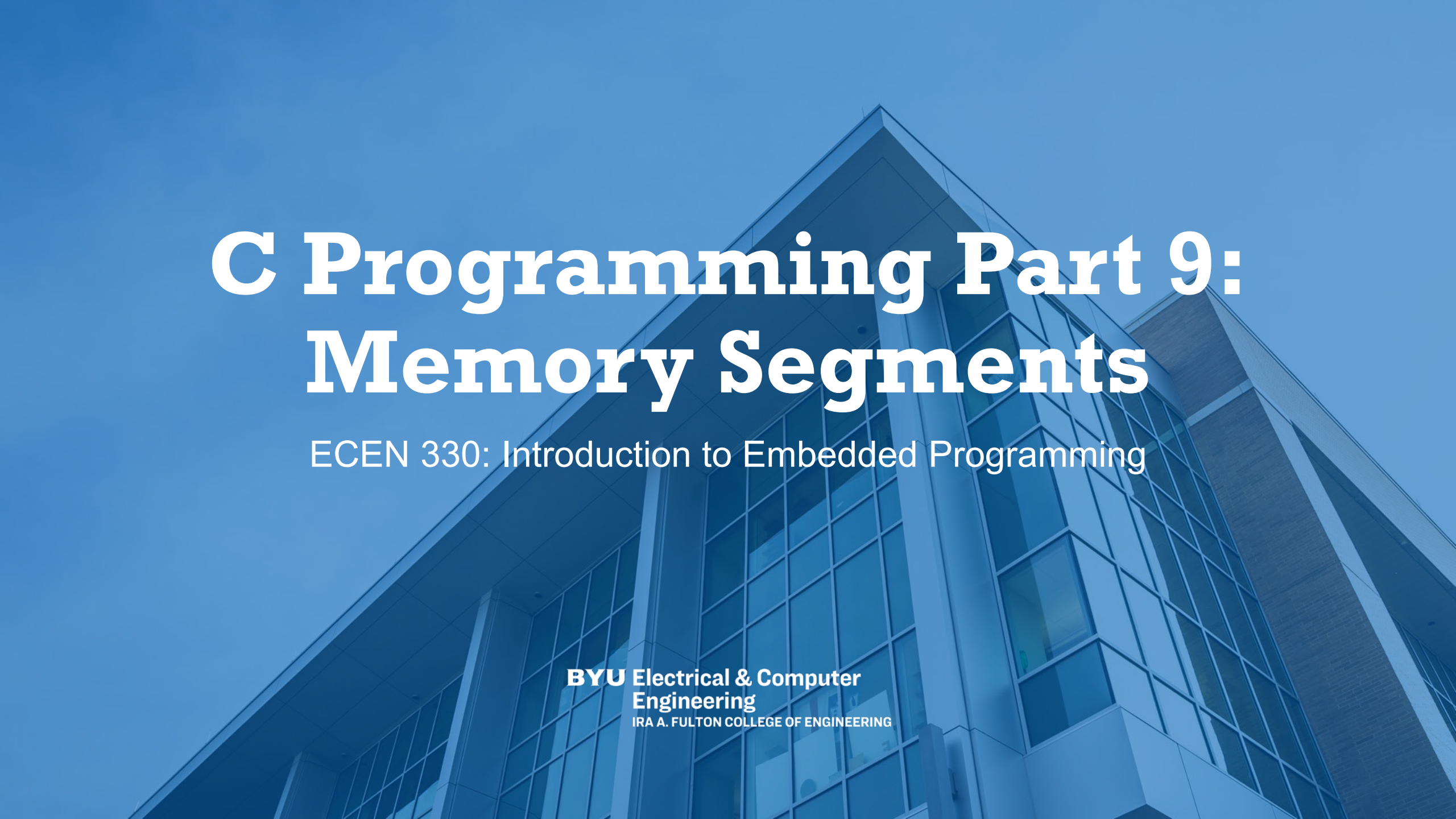


C Programming Part 9: Memory Segments

ECEN 330: Introduction to Embedded Programming

BYU Electrical & Computer Engineering
IRA A. FULTON COLLEGE OF ENGINEERING

Memory

On a bare-metal system (ie no operating system), the memory is divided into several segments.

In general, this consists of:

1. Program instructions
2. Globals
3. Stack
4. Heap

1. Program Instructions

- Your C code is compiled into binary machine instructions.

- Like your variables, these instructions are loaded into memory.

- On some systems this is the same physical memory as the data.
- On some systems this is a separate (read-only) physical memory.

2. Globals

What uses global memory space?

- Global variables, structs, arrays, etc
- Global const variables
- Read-only strings
- Static function variables

Allocation/Deallocation (Lifetime):

- Entire life of the program

Runtime Cost:

- Zero runtime (allocated when your program is compiled)

Size:

- Must be determined at runtime

3. Stack

What uses stack memory space?

- Local/automatic variables
- Function arguments

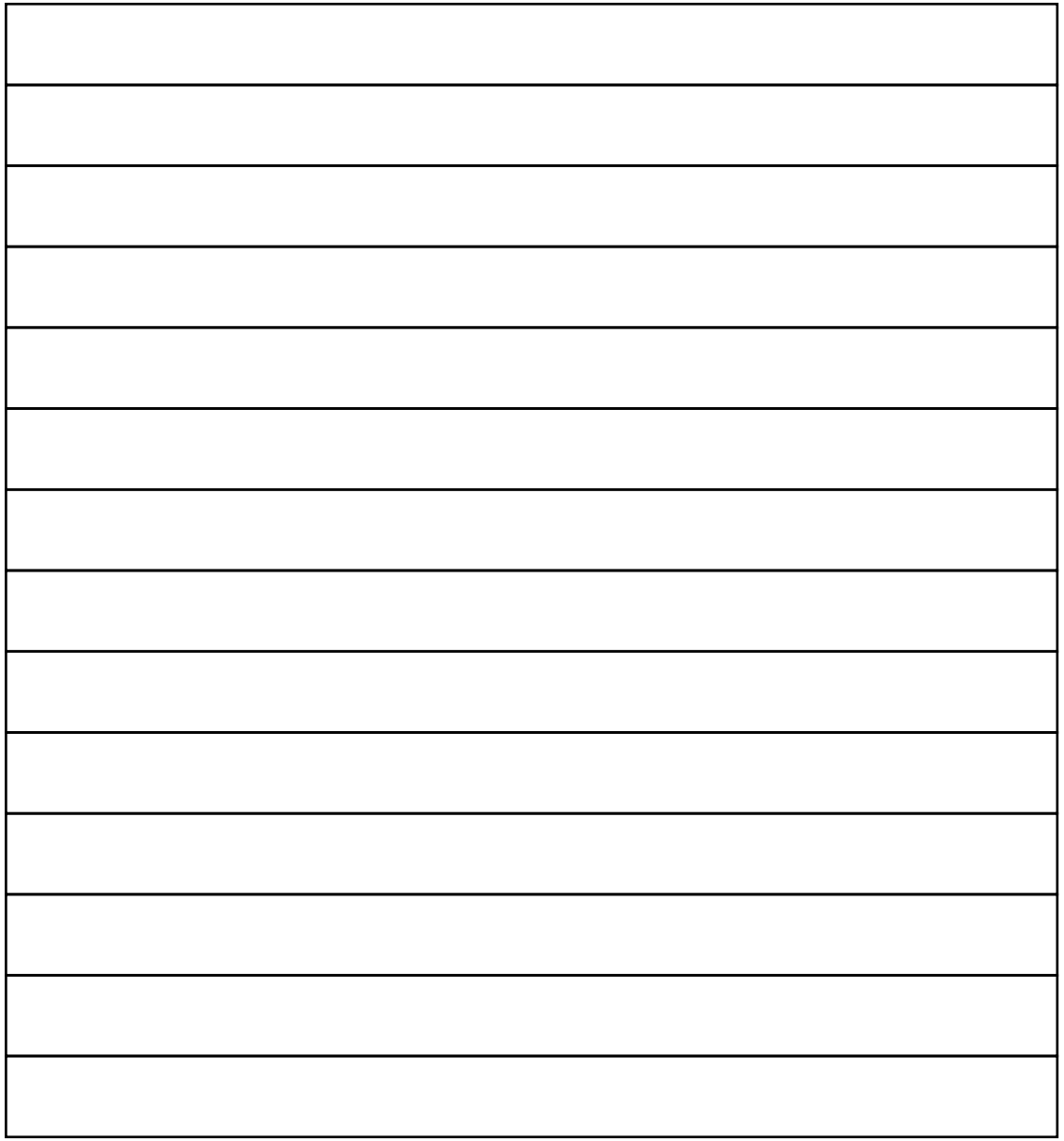
- **Allocation/Deallocation (Lifetime):**
  - Allocated at function start, deallocated at function end

- **Runtime Cost:**
  - Very small runtime (this is the overhead of calling a function, which is very small)

- **Size:**
  - At compile time you have to decide what variables you need and their sizes for each function (**static)**
    - Only exception: Modern C allows local arrays to have variable size.
  - Total stack size can't be determined at compile time.
    - Need to know which functions will be called, how deep recursion will go, etc.

```c
int square(int a) {
    return a * a;
}

int sum_squares(int a, int b) {
    int t1 = square(a);
    int t2 = square(b);
    return t1 + t2;
}

int add3(int c, int d, int e) {
    return c + d + e;
}

int main() {
    int x = 3;
    int y = 5;

    int z = sum_squares(x, y);

    int sum = add3(x, y, z);

    printf("sum: %d\n", sum);
}
```

Stack



4. Heap

What uses heap memory space?

- `malloc()` function returns a pointer to a chunk of heap memory of requested size.

- **Allocation/Deallocation (Lifetime):**
  - Allocated **dynamically** (whenever user calls malloc)
    - This can be done in functions, in loops, in if statements, etc.
  - Allocated forever, until user calls `free()` with the same pointer.

- **Runtime Cost:**
  - Small runtime cost, but larger than the stack
    - malloc needs to find you a piece of memory large enough for your request
    - free needs to return the memory to the pool, and "combine" it with neighboring free chunks
    - Behind the scenes these functions keep a list of allocated and free chunks, their addresses and sizes.

- **Size:**
  - Size not needed (and can't be determined) at compile time. The program can dynamically request as much or as little memory as it needs, and continuously free memory and request more.

Comparison of Data Segments

	Globals	Stack	Heap
Allocation/Deallocation	Automatic	Automatic	Manual
Size	Known at compile time.	Not known at compile time.	Not known at compile time.
Ease of Use	Easy	Easy	Requires more care.
Flexibility	Fixed # of variables and array sizes.	Fixed # of variables, flexible array sizes.	Completely flexible (supports advanced data structures)
Speed	0 time	Very small runtime	Small runtime.