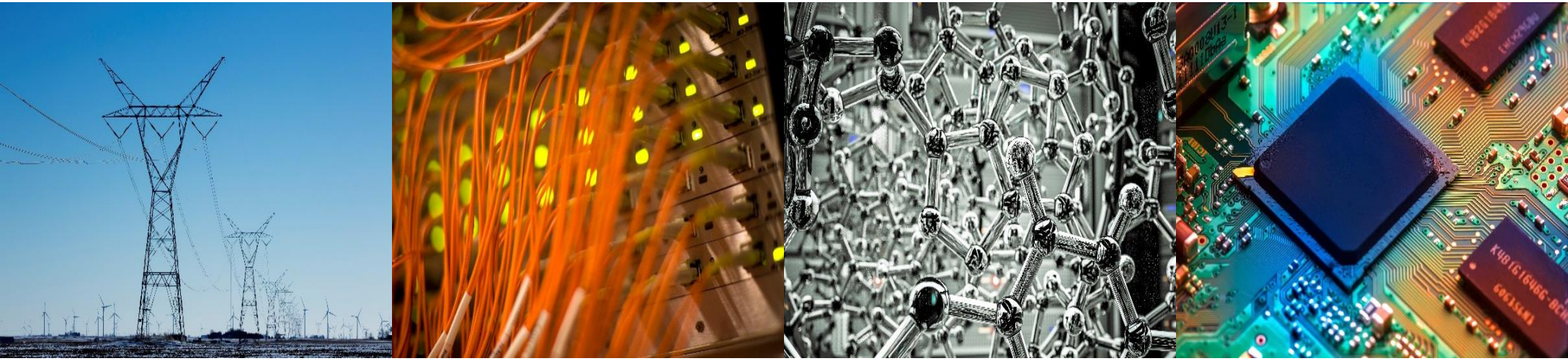
ECE 220 Computer Systems & Programming

Lecture 16 – Dynamic Memory Allocation

October 21, 2025



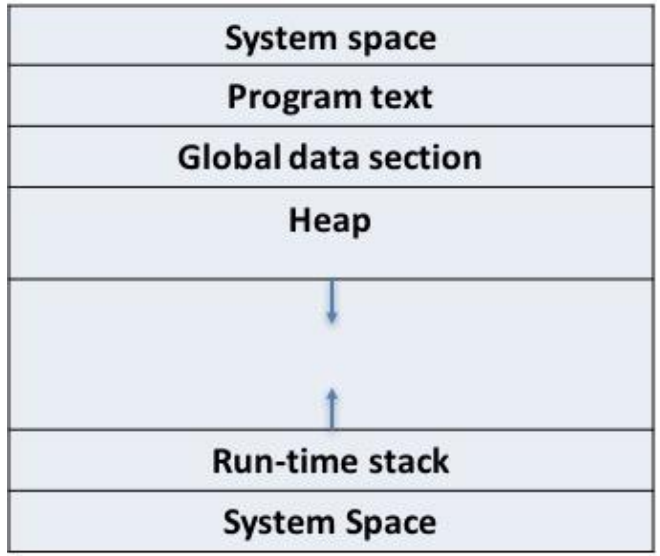
- **DRES-TAC final exam priority deadline: Nov. 1st**

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

Lecture 15 Recap

- Enum
- Struct vs. Union
- Size of struct (padding vs. no padding)
- Access member through a variable vs. through a pointer
- Pointer to struct
- Struct arrays
- Struct pass by value vs. pass by reference
- Struct within a Struct

“Static” vs. Dynamic Memory Allocation



	“Static”	Dynamic
Mechanism of allocation		
Lifetime of memory		
Location of memory		
Size of allocation		

malloc & free

```
void *malloc(size_t size);
```

- allocates a contiguous region of memory on the heap
- size of allocated memory block is indicated by the argument
- returns a generic pointer (of type void *) to the memory, or NULL in case of failure
- allocated memory is not clear (there could be left over junk data!)

```
void free(void *ptr);
```

- frees the block of memory pointed to by ptr
- ptr must be returned by malloc() family of functions

Examples using malloc & free

```
int *ptr = (int *)malloc(sizeof(int));
if(ptr == NULL){
      printf("ERROR - malloc failure!");
      return 1;}
*ptr = 10;
free(ptr);
```

- How can we dynamically allocate space for an integer array with 10 elements?

- What is happening in this block of code?

```
int *ptr = (int *)malloc(sizeof(int));
*ptr = 5;
int *ptr_2 = (int *)malloc(sizeof(int));
*ptr_2 = 6;
ptr = ptr_2;
```

calloc & realloc

```
void *calloc(size_t n_items, size_t item_size);
```

- similar to malloc(), also sets allocated memory to zero
- n_item: the number of items to be allocated, item_size: the size of each item
  → total size of allocated memory = n_items * item_size

```
void *realloc(void *ptr, size_t size);
```

- reallocate memory block to a different size (change the size of memory block pointed to by ptr)
- returns a pointer to the newly allocated memory block (it may be changed)
- Unless ptr == NULL, it must be returned by the malloc() family of functions
- if ptr == NULL → same as malloc()
- if size == 0, ptr != NULL → implementation specific

Possible scenarios when using realloc

1. from a zero sized block to a non-zero sized block → **same as malloc**
2. from a non-zero sized block to a zero sized block → **implementation specific**
3. from a larger block to a smaller block
4. from a smaller block to a larger block

Examples using calloc & realloc

- What does this block of code do?

```
char *ptr2 = (char *)calloc(100, sizeof(char));
if(ptr2 == NULL){
   printf("ERROR – calloc failure!");
   return 1;}
strncpy(ptr2, "Example using calloc", 100);
```

- What is happening now?

```
char *ptr3 = (char *)realloc(ptr2, 200*sizeof(char));
if(ptr3 == NULL){
   printf("ERROR – realloc failure!");
   return 1;}
```

- How much memory are we deallocating here?

```
free(ptr3);
```

Exercise:

```
typedef struct studentStruct{
   char *NAME;
   int UIN;
   float GPA;
}student;
```

1. Dynamically allocate memory for 200 student records
2. Initialize name to “To be set”, UIN to -1 and GPA to 0.0 for all 200 records (**hint: you will also need to allocate an array of 100 chars to hold the name for each record**)
3. Add 200 more student records and initialize them as in step 2
4. Free up memory space for all the records