ECE 220 Computer Systems & Programming

Structs & Dynamic Memory Allocation

ECE ILLINOIS

ILLINOIS

Pointer to Struct

```
student ece220[200];
student s1;
student *arr_ptr, *s_ptr;
arr_ptr = ece220; //pointer to a struct array
s_ptr = &s1; //pointer to a struct

strncpy(arr_ptr->Name, “John Doe”, sizeof(s1.Name));
arr_ptr->UIN = 123456789;
arr_ptr->GPA = 3.89;
//which student record has been changed?

arr_ptr++; //where is ptr pointing to now?

//What is the difference between the following function calls?
PrintName(s1);
PrintName(&s1);
```

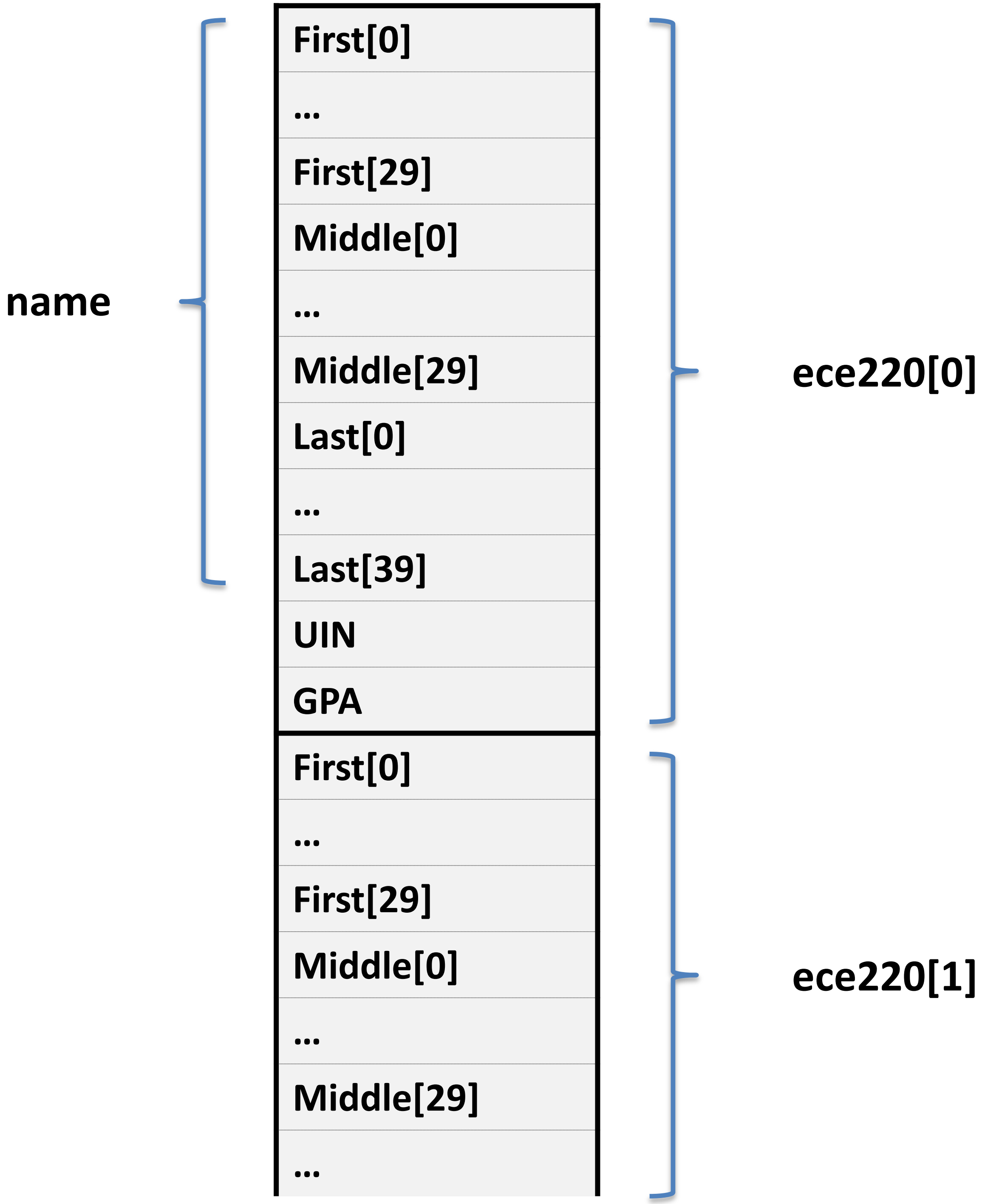
Struct within a Struct

```
typedef struct StudentName
{
    char First[30];
    char Middle[30];
    char Last[40];
}name;
```

```
typedef struct StudentStruct
{
    name Name;
    int UIN;
    float GPA;
}student;
```

```
student ece220[200];
student *ptr;
ptr = ece220;

//How can we set the ‘First’ name in the first student record?
strncpy(                       , “John”,                         );
```



name

First[0]

…

First[29]

Middle[0]

…

Middle[29]

Last[0]

…

Last[39]

UIN

GPA

ece220[0]

First[0]

…

First[29]

Middle[0]

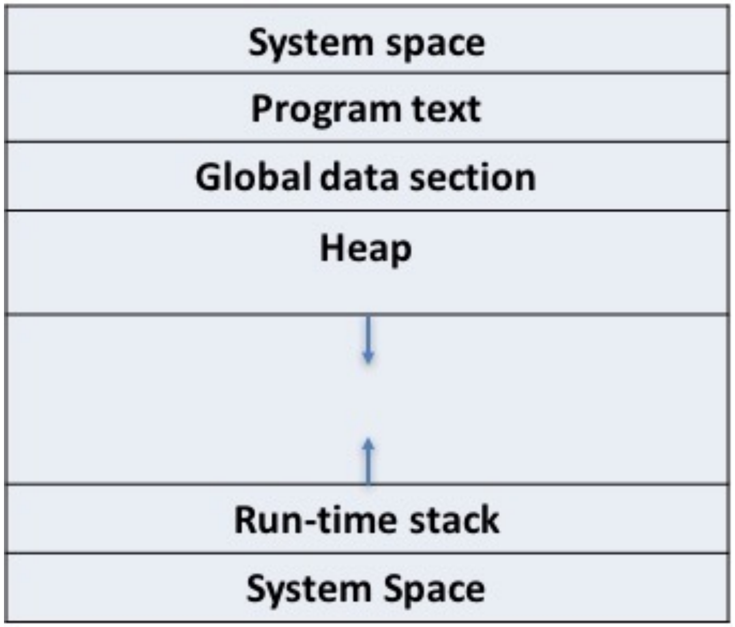
…

Middle[29]

…

ece220[1]

Static vs. Dynamic Memory Allocation



System space

Program text

Global data section

Heap

Run-time stack

System Space

	Static	Dynamic
Mechanism of allocation	Based on Declaration	Explicit Allocation Call
Lifetime of memory	Forever	Until Freed
Location of memory	Global Data	Heap
Size of allocation	Fixed at Compile Time	Chosen at Runtime

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

Example using malloc & free:

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

- How do we dynamically allocate space for an int array with 10 elements?

- What is happening in the block of code below?

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

Exercise:

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

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

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 → same as free()

Example using calloc & realloc:

- What does this block of code do?

```
char *ptr2 = calloc(50, sizeof(char));
if(ptr2 == NULL){
   printf(“ERROR – calloc failure!”);
   return 1;}
strncpy(ptr2, “Example using calloc”, 50);
```

- What happens now?

```
char *ptr3 = realloc(ptr2, 100*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 (hint: you will also need to allocate an array of 100 chars to hold the name for each record)
2. Initialize name to “To be set”, UIN to -1 and GPA to 0.0 for all 200 records
3. Add 200 more student records and initialize them as in step 2
4. Free up memory space for all the records

Common Mistakes in Dynamic Memory Allocation

- Dangling Pointers
- Memory Leaks
- Accessing Beyond Bounds

Dynamic Memory Allocation Summary

- Static vs. Dynamic allocation
- Functions: malloc(), free(), calloc(), realloc()
- Memory Leak vs. Segmentation Fault