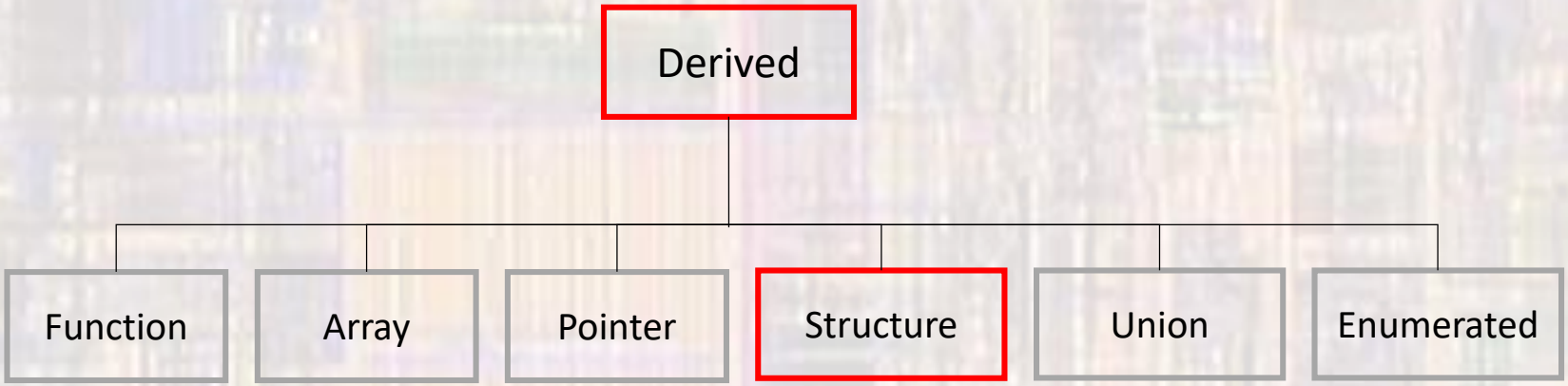
Structures

Last updated 1/15/19

Structures

- C / C++ Types



Structures

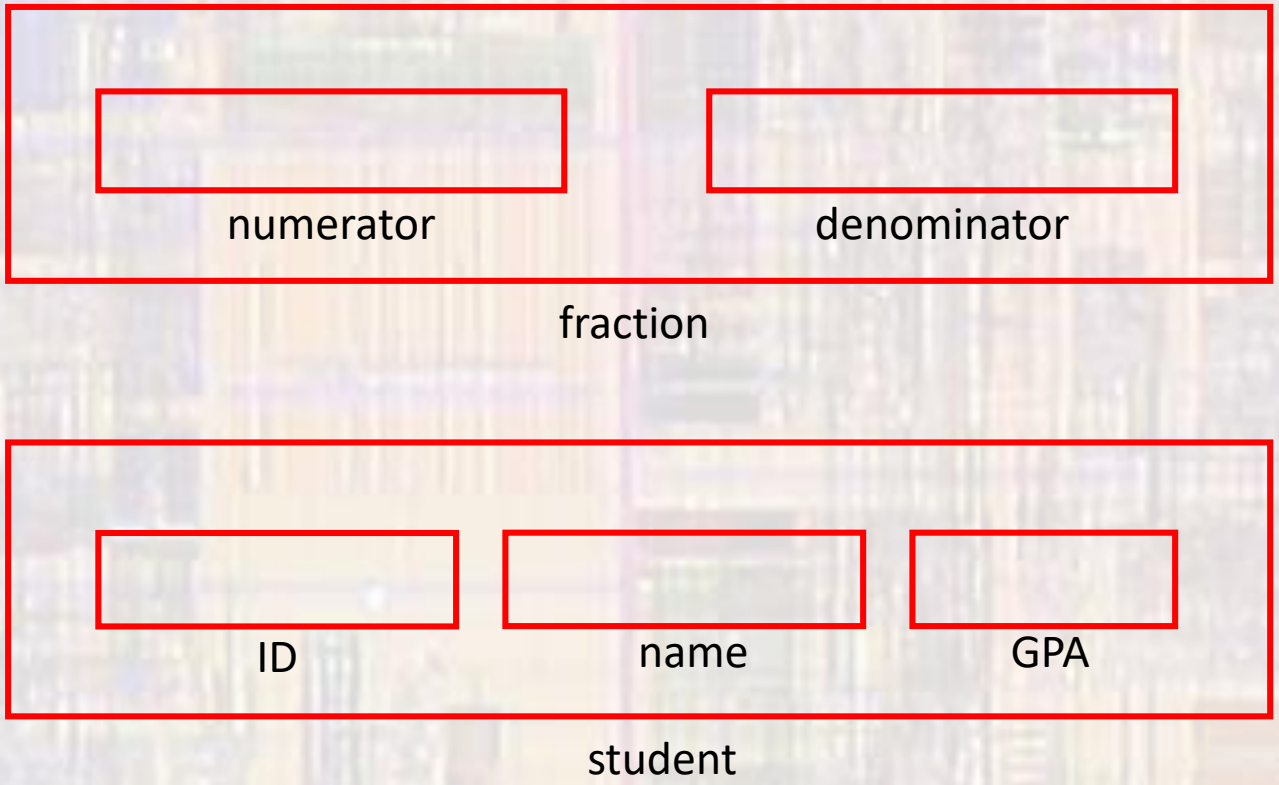
- Structure
  - Collection of related elements
  - Not necessarily the same type
  - Sharing a single name

Structures

- Structure
  - Elemental unit is called a Field
  - Looks just like a variable
    - has a type
    - takes up memory space
    - can be assigned values
    - can be read
  - Only difference is that a Field is part of a Structure

Structures

- Structure



Structures

- Structure
  - TypeDef definition
    - defines a new type
    - new elements of the structure type can be created via declaration

```
typedef struct{
  field list
} TYPE;


typedef struct{
  char ID[10];
  char name[26];
  float GPA;
} STUDENT;                          // type definition name
```

Structures

- Structure
  - TypeDef definition

```
typedef struct{
   char ID[10];
   char name[26];
   float GPA;
} STUDENT;                          // type definition name

…
int foo;
STUDENT student1;
STUDENT student2;
```

Structures

- Structure
  - TypeDef definition
    - Initialization

…

int foo;

STUDENT student1 = {12A56B, Joe Smith, 3.45};

STUDENT student2 = {12A56B, Joe Smith};

GPA defaults to 0.0

If not specified, int and float fields default to 0, 0.0

char fields default to null - \0

Structures

- Structure
  - Access a field in a structure

structure.field

student1.ID
student1.name
student1.GPA

Structures

- Structure
  - Access a field in a structure

```
student2.GPA = 2.5;

if(student1.GPA >= 3.5){
   …
}

printf("student GPA: %.2f", student2.GPA);

scanf("%f", &student1.GPA);
```

Structures

- Structure
  - Manipulation
    - Only one operation – assignment
    - Make a complete copy

```
student2 = student1;
```

Structures

- Structure
  - Pointers to structures

```
typedef struct{
  char ID[10];
  char name[26];
  float GPA;
} STUDENT;
```

```
STUDENT* student_ptr;          // define a pointer of STUDENT type

student_ptr = &student1;       // student_ptr now points to student1
```

Structures

- Structure
  - Pointers to structures – accessing fields

```
STUDENT* student_ptr;          // define a pointer of STUDENT type

student_ptr = &student1;       // student_ptr now points to student1


(*student_ptr).GPA = 3.66;     // dereference

student_ptr->GPA = 3.66;       // indirect selection
```

Structures

```
#include <stdio.h>
#include <unistd.h>

// Type definitions
typedef struct{     // define a type: CLOCK
   int hr;
   int min;
   int sec;
} CLOCK;

// Function prototypes
void increment (CLOCK* clock_struct);
void display (CLOCK clock_st);

int main(void){
  setbuf(stdout, NULL);  // disable buffering

  //  Local variables
  CLOCK clock1 = {11, 59, 57};

  //  Operation
  for(; ; ){
     increment(&clock1);
     display(clock1);
     sleep(1);
  }
  return 0;
}
```

Pointer to structure
pointer notation for fields

```
void increment(CLOCK* clock)_struct{
  (clock_struct->sec)++;        // increment seconds
   if (clock_struct->sec == 60){
      clock_struct->sec = 0;
      (clock_struct->min)++;     // increment minutes
      if(clock-_struct>min == 60){
         clock_struct->min = 0;
         (clock_struct->hr)++;
         if(clock_struct->hr == 12){
            clock_struct->hr = 0;
         } // end if hr
      } // end if min
   } // end if sec
   return;
}
```

structure passed
structure notation for fields

```
void display(CLOCK clock_st){
   printf("%02d:%02d:%02d\n", clock_st.hr, clock_st.min, clock_st.sec);
   return;
}
```

```
11:59:56
11:59:57
11:59:58
11:59:59
00:00:00
00:00:01
00:00:02
```