Structures

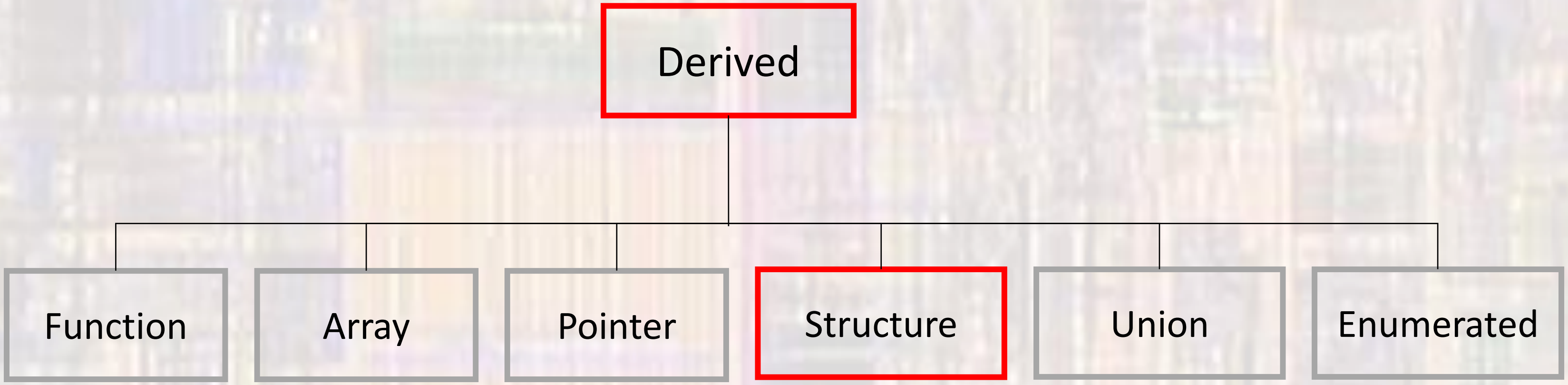
Last updated 10/29/20

Structures

- These slides introduce the c type structure
- Upon completion: You should be able interpret and code using structures

Type Definition

- C Types



Structures

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

Structures

- Members

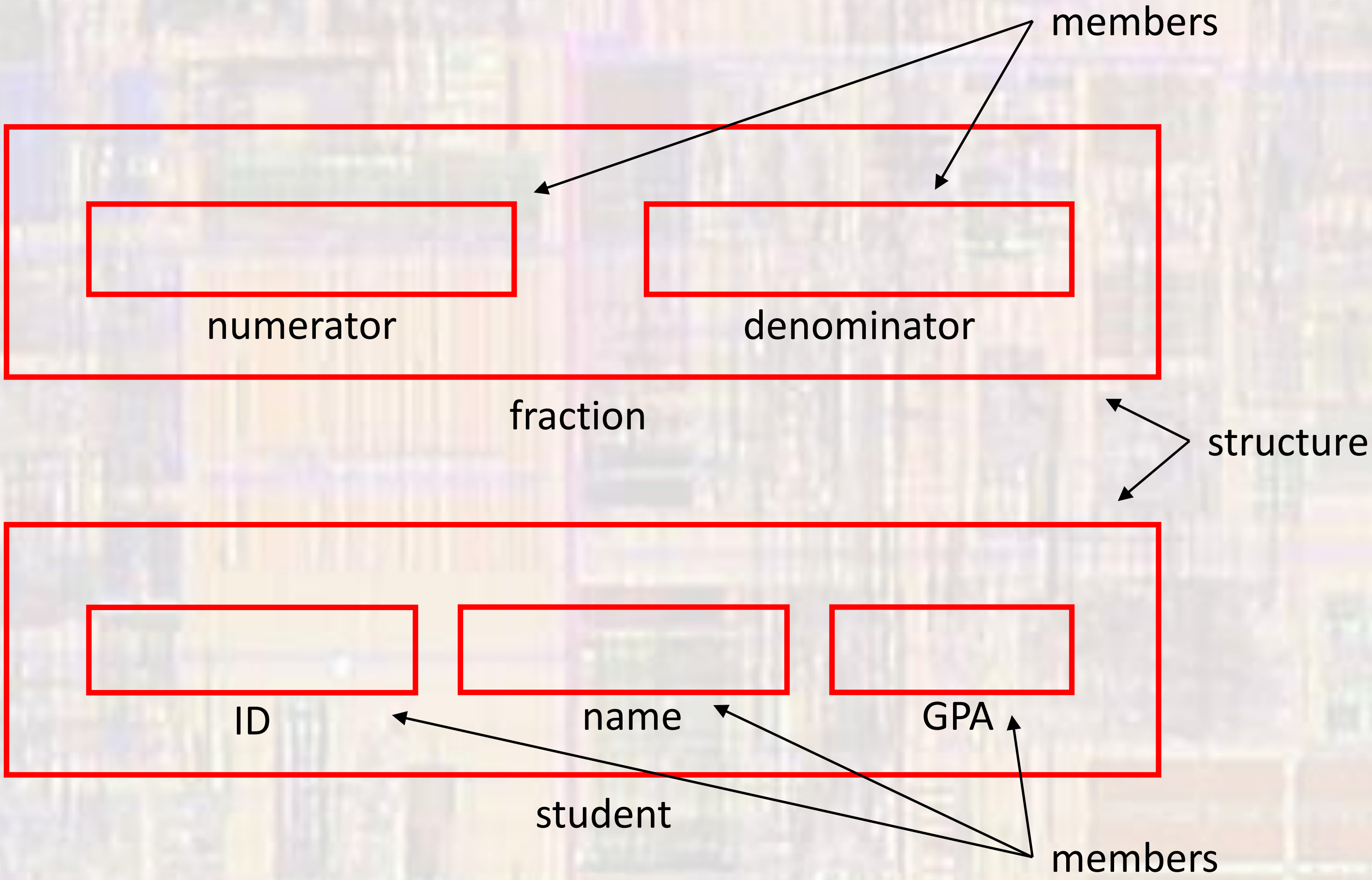
  - Elemental unit is called a Member (Field)

  - Members look just like a variable
    - have a type
    - takes up memory space
    - can be assigned values
    - can be read

  - Only difference is that a Member is part of a Structure

Structures

- Members



Structures

- 3 ways to create structures
  - Individual

format

```
struct {
   list of members
} variable name(s);
```

declaration

```
struct {
   int id;
   char name[26];
   float gpa;
} stu1, stu2;
```

All members are unknown values

Structure variables
stu1, stu2

declaration/initialization

```
struct {
   int id;
   char name[26];
   float gpa;
} stu1 = {.id=245, .name="john", .gpa=3.5},
   stu2 = {246, "sally", 3.6};
```

Order doesn't matter

Order matters

If not fully specified, int and float members default to 0, 0.0
char members default to null - \0

Structures

- 3 ways to create structures
  - Tag

format

```
struct tag{
   list of members
} ;
```

definition

```
struct student{
   int id;
   char name[26];
   float gpa;
} ;
```

Structure variables
stu0, stu1, stu2

declaration

```
struct student stu0;
```

All members are unknown values

Order doesn't matter

declaration/initialization

```
struct student stu1 = {.id=245,
                       .name="john",
                       .gpa=3.5};
```

If not fully specified, int and float members default to 0, 0.0
char members default to null - \0

declaration/ initialization

```
struct student stu1 = {245,
                       "john",
                       3.5};
```

Order matters

Structures

- 3 ways to create structures
  - Typedef – create a new type

format

```
typedef struct {
   list of members
} type_name ;
```

definition

```
typedef struct {
   int id;
   char name[26];
   float gpa;
} student ;
```

Structure variables
stu0, stu1, stu2

declaration

```
student stu0;
```

All members are unknown values

Order doesn't matter

declaration/initialization

```
student stu1 = {.id=245,
                .name=“john”,
                .gpa=3.5};
```

If not fully specified, int and float members default to 0, 0.0
char members default to null - \0

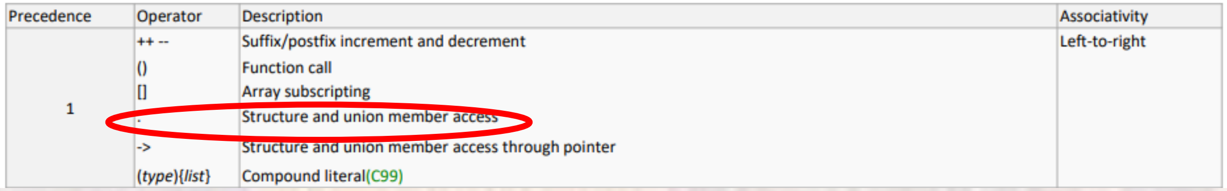
declaration/ initialization

```
student stu1 = {245,
                “john”,
                3.5};
```

Order matters

Structures

- Member Access
  - You can access the member variables using the structure access operator
  - structure access operator .



Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){*list*}	Compound literal(C99)	

structure_variable.member

Given a structure variable named stu1

stu1.id

stu1.name

stu1.gpa

Structures

- Member Access

```
stu2.gpa = 2.5;          // set the member variable gpa to 2.5

if(stu1.gpa >= 3.5){
   …
}

printf(“student GPA: %.2f”, stu2.gpa);

scanf(“%f”, &stu1.gpa);
```

Note: access operator . has higher priority than address-of operator & so no parenthesis required

Structures

- Structure
    - Manipulation
        - Only one operation – assignment

```
stu2 = stu1;                    // copy all member values from stu1 to stu2
                                // must be the same structure (or type)
```

Structures

- Pointers and structures
  - Given a structure variable created using one of the 3 processes
  - Can create and use structure pointers

Given structure variable stu1 of structure type student

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

student_ptr = &stu1;            // student_ptr now points to stu1
```

  - All normal pointer operations can be applied

(Note: pointer arithmetic operates on the entire structure, not on the elements)

Structures

- Pointers and structures
  - 2 ways to access a member value from a pointer

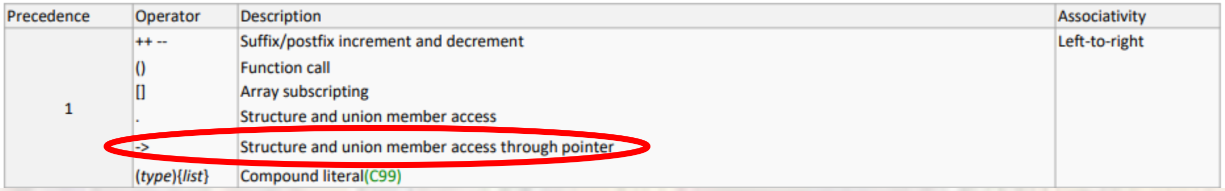
Given structure variable stu1 of structure type student

```
student* student_ptr;          // define a pointer of student type
student_ptr = &stu1;           // student_ptr now points to stu1
```

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

Note () required to ensure the structure is dereferenced before accessing the member

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



Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){*list*}	Compound literal(C99)	

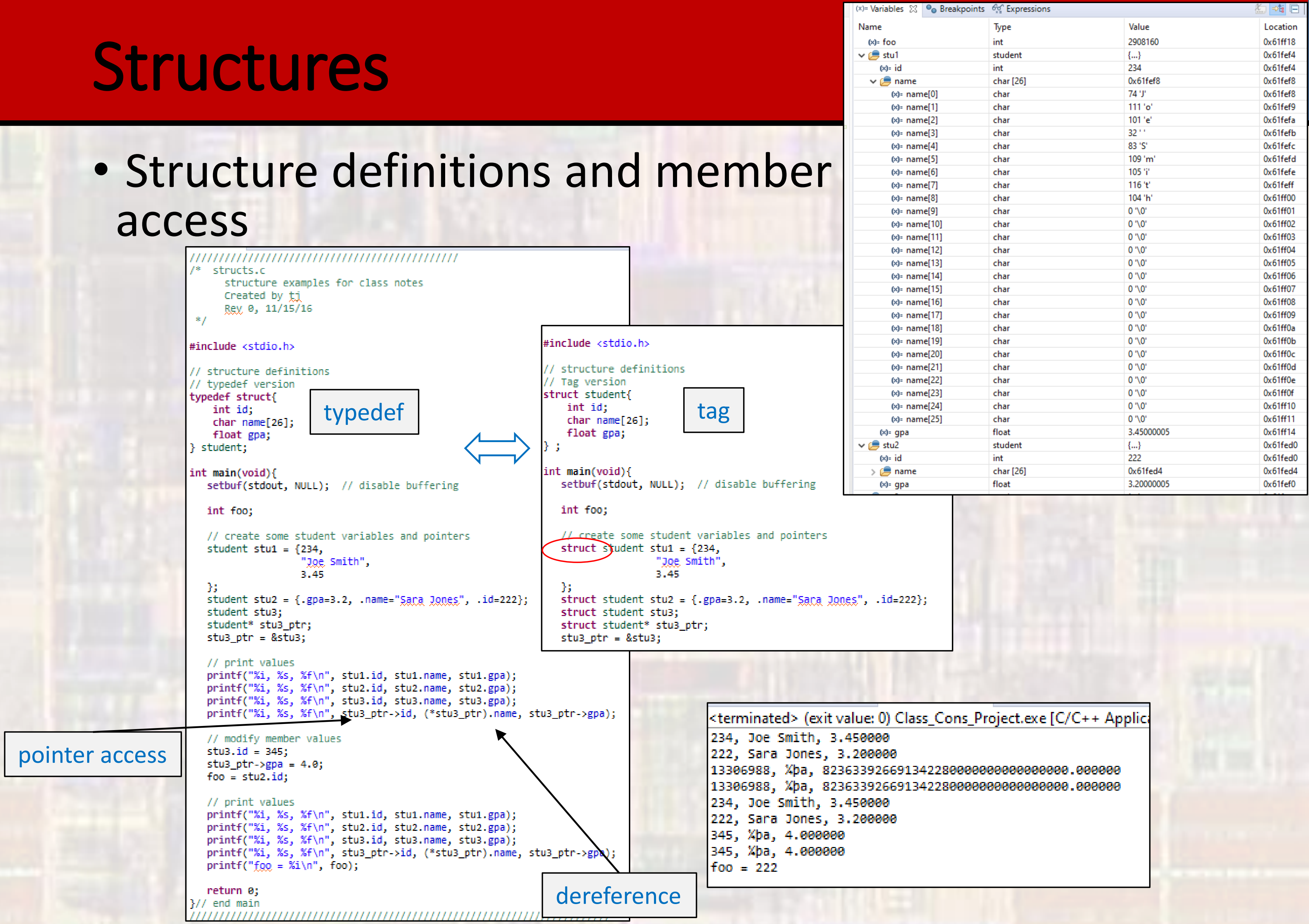
Structures

- Scope considerations
  - Structures are treated like any other variable with respect to scope
  - Structure members are considered to be in the structure scope
    - No conflict in having structure member names the same as other variables since their scope is limited to the structure
  - Typedef and Tag definitions typically belong in the global section of a file – so everything recognizes them
  - Variable declarations are treated like any other variable
    - Place them in whatever scope is appropriate

  - We will use either Typedef or Tag definitions to avoid issues with Individual definitions and scope

Structures

- Structure definitions and member access



```
////////////////////////////////////////////////
/*  structs.c
      structure examples for class notes
      Created by tj
      Rev 0, 11/15/16
 */

#include <stdio.h>

// structure definitions
// typedef version
typedef struct{
    int id;
    char name[26];
    float gpa;
} student;

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

   int foo;

   // create some student variables and pointers
   student stu1 = {234,
                   "Joe Smith",
                   3.45
   };
   student stu2 = {.gpa=3.2, .name="Sara Jones", .id=222};
   student stu3;
   student* stu3_ptr;
   stu3_ptr = &stu3;

   // print values
   printf("%i, %s, %f\n", stu1.id, stu1.name, stu1.gpa);
   printf("%i, %s, %f\n", stu2.id, stu2.name, stu2.gpa);
   printf("%i, %s, %f\n", stu3.id, stu3.name, stu3.gpa);
   printf("%i, %s, %f\n", stu3_ptr->id, (*stu3_ptr).name, stu3_ptr->gpa);

   // modify member values
   stu3.id = 345;
   stu3_ptr->gpa = 4.0;
   foo = stu2.id;

   // print values
   printf("%i, %s, %f\n", stu1.id, stu1.name, stu1.gpa);
   printf("%i, %s, %f\n", stu2.id, stu2.name, stu2.gpa);
   printf("%i, %s, %f\n", stu3.id, stu3.name, stu3.gpa);
   printf("%i, %s, %f\n", stu3_ptr->id, (*stu3_ptr).name, stu3_ptr->gpa);
   printf("foo = %i\n", foo);

   return 0;
}// end main
////////////////////////////////////////////////
```

typedef

pointer access

dereference

```
#include <stdio.h>

// structure definitions
// Tag version
struct student{
    int id;
    char name[26];
    float gpa;
} ;

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

   int foo;

   // create some student variables and pointers
   struct student stu1 = {234,
                   "Joe Smith",
                   3.45
   };
   struct student stu2 = {.gpa=3.2, .name="Sara Jones", .id=222};
   struct student stu3;
   struct student* stu3_ptr;
   stu3_ptr = &stu3;
```

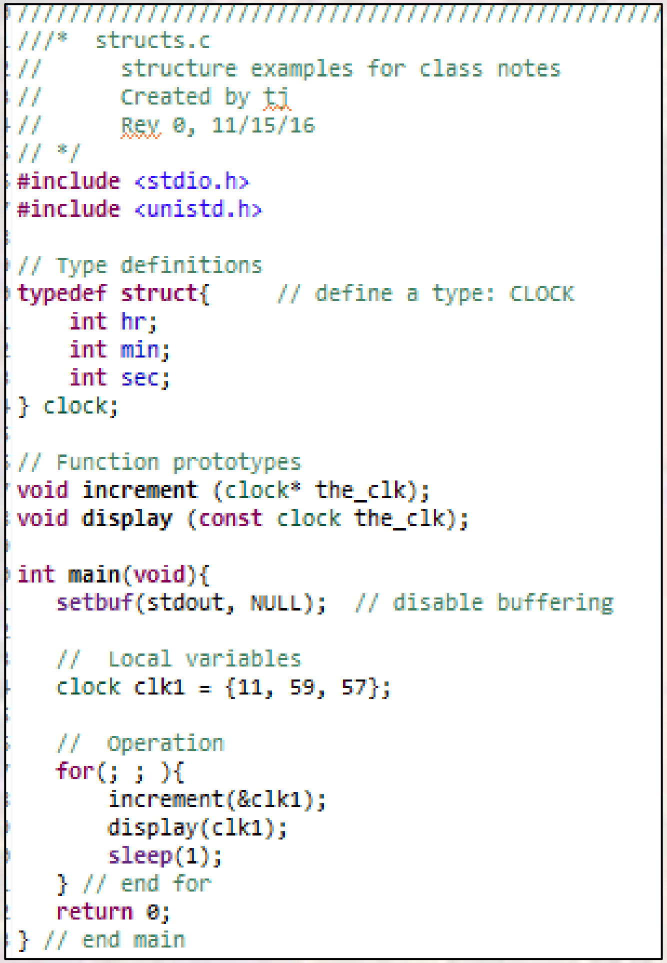
tag

Name	Type	Value	Location
foo	int	2908160	0x61ff18
stu1	student	{...}	0x61fef4
id	int	234	0x61fef4
name	char [26]	0x61fef8	0x61fef8
name[0]	char	74 'J'	0x61fef8
name[1]	char	111 'o'	0x61fef9
name[2]	char	101 'e'	0x61fefa
name[3]	char	32 ' '	0x61fefb
name[4]	char	83 'S'	0x61fefc
name[5]	char	109 'm'	0x61fefd
name[6]	char	105 'i'	0x61fefe
name[7]	char	116 't'	0x61feff
name[8]	char	104 'h'	0x61ff00
name[9]	char	0 '\0'	0x61ff01
name[10]	char	0 '\0'	0x61ff02
name[11]	char	0 '\0'	0x61ff03
name[12]	char	0 '\0'	0x61ff04
name[13]	char	0 '\0'	0x61ff05
name[14]	char	0 '\0'	0x61ff06
name[15]	char	0 '\0'	0x61ff07
name[16]	char	0 '\0'	0x61ff08
name[17]	char	0 '\0'	0x61ff09
name[18]	char	0 '\0'	0x61ff0a
name[19]	char	0 '\0'	0x61ff0b
name[20]	char	0 '\0'	0x61ff0c
name[21]	char	0 '\0'	0x61ff0d
name[22]	char	0 '\0'	0x61ff0e
name[23]	char	0 '\0'	0x61ff0f
name[24]	char	0 '\0'	0x61ff10
name[25]	char	0 '\0'	0x61ff11
gpa	float	3.45000005	0x61ff14
stu2	student	{...}	0x61fed0
id	int	222	0x61fed0
name	char [26]	0x61fed4	0x61fed4
gpa	float	3.20000005	0x61fef0

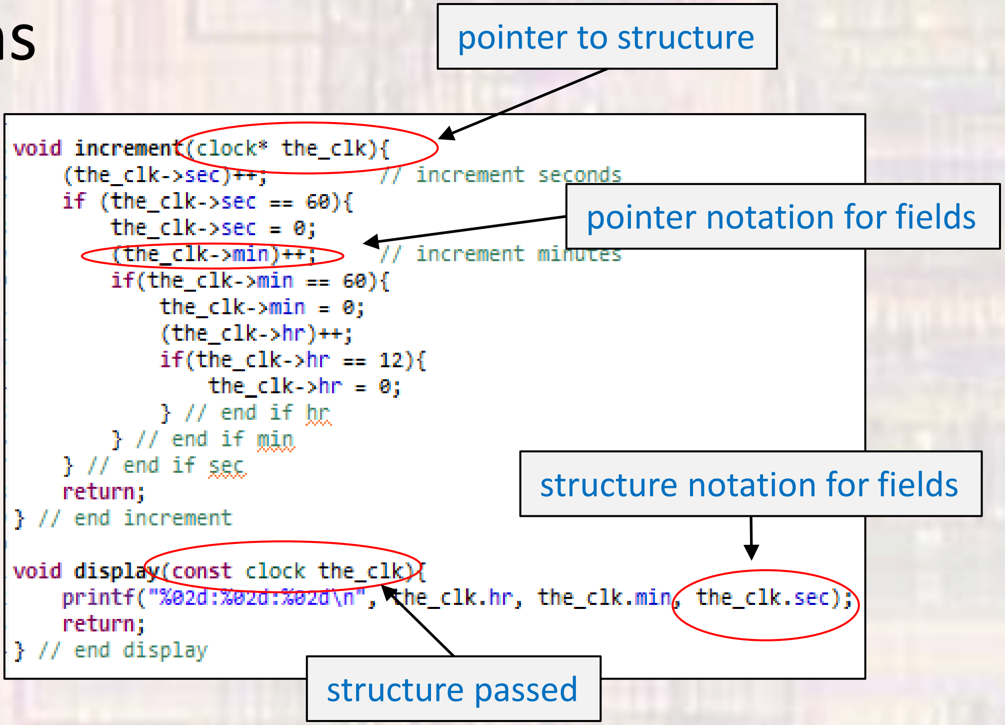
```
<terminated> (exit value: 0) Class_Cons_Project.exe [C/C++ Applic
234, Joe Smith, 3.450000
222, Sara Jones, 3.200000
13306988, ¾þa, 823633926691342280000000000000000.000000
13306988, ¾þa, 823633926691342280000000000000000.000000
234, Joe Smith, 3.450000
222, Sara Jones, 3.200000
345, ¾þa, 4.000000
345, ¾þa, 4.000000
foo = 222
```

Structures

- Structures and functions



```
/////////////////////////////////////////////////////////
///*  structs.c
//      structure examples for class notes
//      Created by tj
//      Rev 0, 11/15/16
// */
#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* the_clk);
void display (const clock the_clk);

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

   //  Local variables
   clock clk1 = {11, 59, 57};

   //  Operation
   for(; ; ){
       increment(&clk1);
       display(clk1);
       sleep(1);
   } // end for
   return 0;
} // end main
```



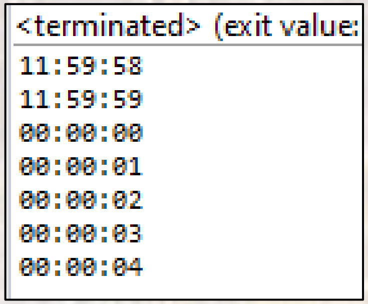
```
void increment(clock* the_clk){
    (the_clk->sec)++;           // increment seconds
    if (the_clk->sec == 60){
        the_clk->sec = 0;
        (the_clk->min)++;       // increment minutes
        if(the_clk->min == 60){
            the_clk->min = 0;
            (the_clk->hr)++;
            if(the_clk->hr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

void display(const clock the_clk){
    printf("%02d:%02d:%02d\n", the_clk.hr, the_clk.min, the_clk.sec);
    return;
} // end display
```

pointer to structure

pointer notation for fields

structure notation for fields

structure passed

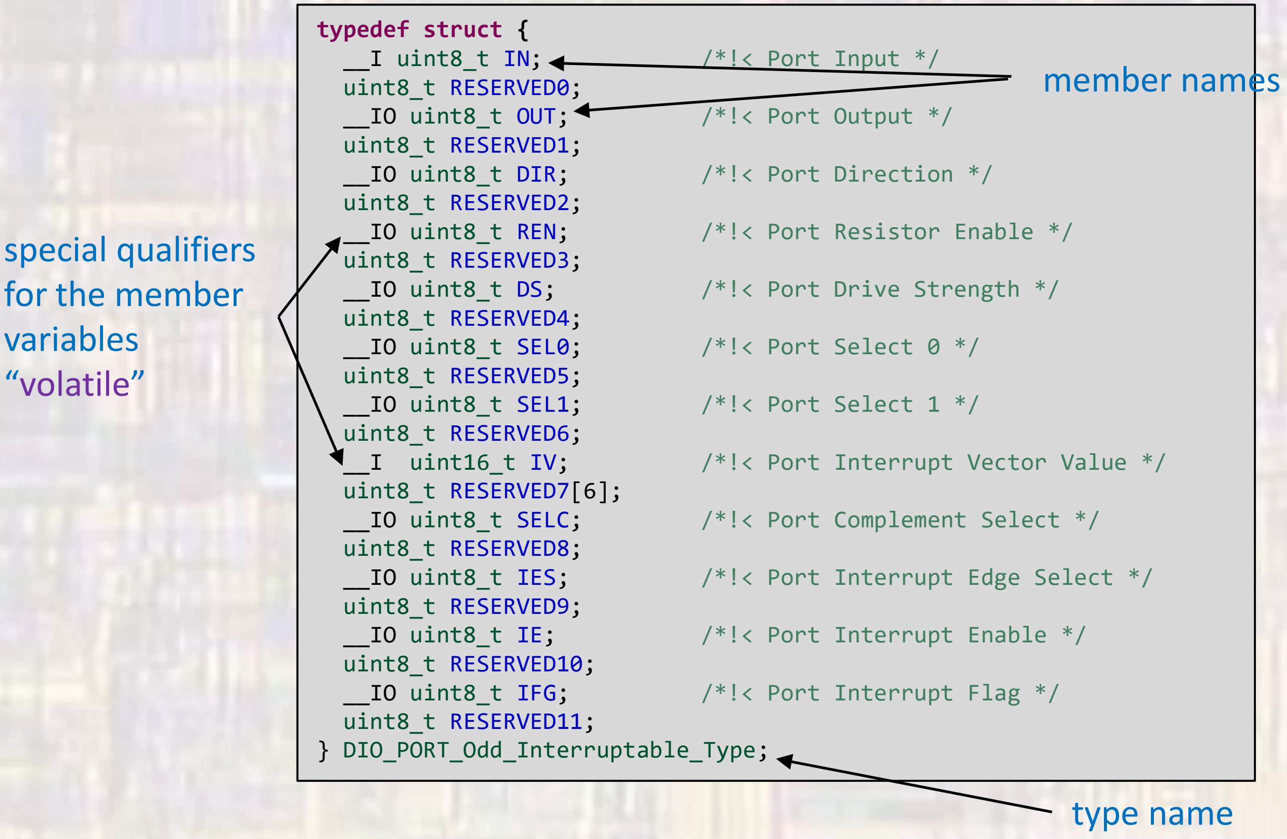


```
<terminated> (exit value:
11:59:58
11:59:59
00:00:00
00:00:01
00:00:02
00:00:03
00:00:04
```

Structures

- Register Access – revisited
  - MSP registers are defined as structures

Port register structure



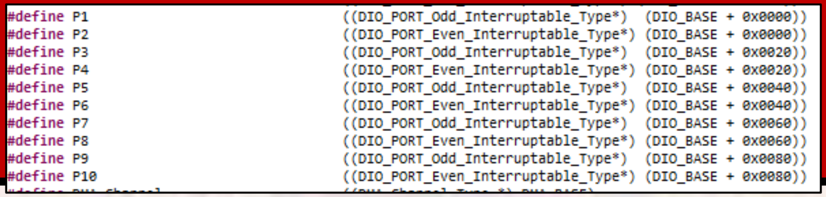
```
typedef struct {
  __I uint8_t IN;            /*!< Port Input */
  uint8_t RESERVED0;
  __IO uint8_t OUT;          /*!< Port Output */
  uint8_t RESERVED1;
  __IO uint8_t DIR;          /*!< Port Direction */
  uint8_t RESERVED2;
  __IO uint8_t REN;          /*!< Port Resistor Enable */
  uint8_t RESERVED3;
  __IO uint8_t DS;           /*!< Port Drive Strength */
  uint8_t RESERVED4;
  __IO uint8_t SEL0;         /*!< Port Select 0 */
  uint8_t RESERVED5;
  __IO uint8_t SEL1;         /*!< Port Select 1 */
  uint8_t RESERVED6;
  __I  uint16_t IV;          /*!< Port Interrupt Vector Value */
  uint8_t RESERVED7[6];
  __IO uint8_t SELC;         /*!< Port Complement Select */
  uint8_t RESERVED8;
  __IO uint8_t IES;          /*!< Port Interrupt Edge Select */
  uint8_t RESERVED9;
  __IO uint8_t IE;           /*!< Port Interrupt Enable */
  uint8_t RESERVED10;
  __IO uint8_t IFG;          /*!< Port Interrupt Flag */
  uint8_t RESERVED11;
} DIO_PORT_Odd_Interruptable_Type;
```

member names

special qualifiers for the member variables "volatile"

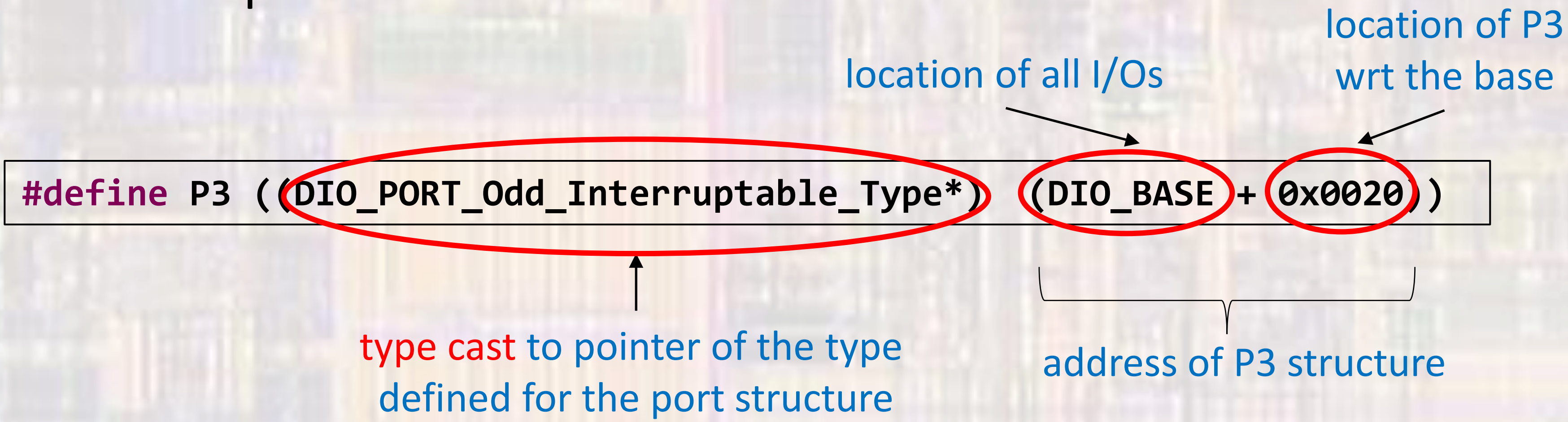
type name

Structures



```
#define P1                                 ((DIO_PORT_Odd_Interruptable_Type*)  (DIO_BASE + 0x0000))
#define P2                                 ((DIO_PORT_Even_Interruptable_Type*) (DIO_BASE + 0x0000))
#define P3                                 ((DIO_PORT_Odd_Interruptable_Type*)  (DIO_BASE + 0x0020))
#define P4                                 ((DIO_PORT_Even_Interruptable_Type*) (DIO_BASE + 0x0020))
#define P5                                 ((DIO_PORT_Odd_Interruptable_Type*)  (DIO_BASE + 0x0040))
#define P6                                 ((DIO_PORT_Even_Interruptable_Type*) (DIO_BASE + 0x0040))
#define P7                                 ((DIO_PORT_Odd_Interruptable_Type*)  (DIO_BASE + 0x0060))
#define P8                                 ((DIO_PORT_Even_Interruptable_Type*) (DIO_BASE + 0x0060))
#define P9                                 ((DIO_PORT_Odd_Interruptable_Type*)  (DIO_BASE + 0x0080))
#define P10                                ((DIO_PORT_Even_Interruptable_Type*) (DIO_BASE + 0x0080))
```

- Register Access – revisited
  - MSP registers are defined as structures
  - “msp.h” includes a series of #define statements



```
#define P3 ((DIO_PORT_Odd_Interruptable_Type*) (DIO_BASE + 0x0020))
```

location of all I/Os

location of P3 wrt the base

type cast to pointer of the type defined for the port structure

address of P3 structure

P3 is now defined as the address (pointer of the port structure type) pointing to the beginning of the Port 3 structure

Structures

- Register Access – revisited
  - MSP registers are defined as structures
  - “msp.h” includes a series of #define statements
  - Port structure members are accessed using the structure pointer access operator ->

```
P3->DIR = P3->DIR | 0x04;
```

Dereferences the P3 pointer to access the DIR member

We could also write

```
(*P3).DIR = (*P3).DIR | 0x04;
```