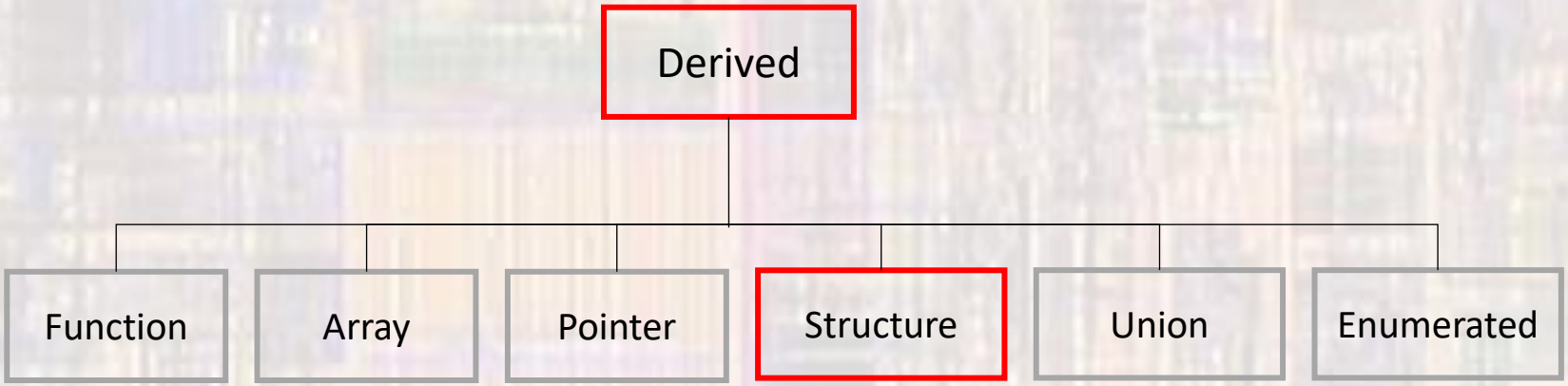
Structures

Last updated 6/22/23

These slides introduce structures in C

Structures

- C Types



Structures

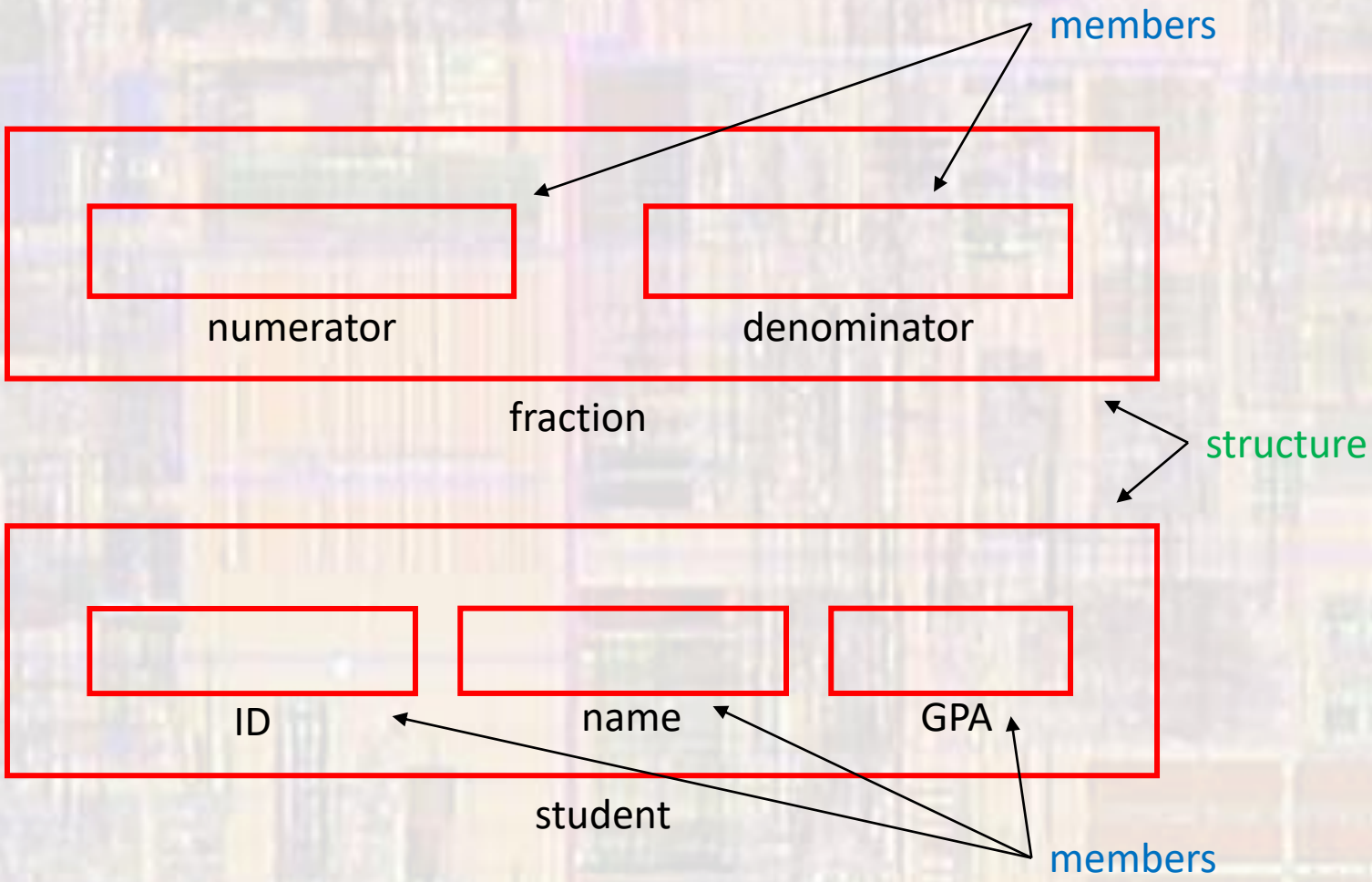
- Concept
  - Collection of related elements
  - Not necessarily the same type
  - Sharing a single name

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 structures declared

format

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

declaration

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

All members values are unknown

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

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

Order matters

Structures

- 3 ways to create structures
  - Tag – like a new type (struct student)

format

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

definition

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

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 ;
```

declaration
```
student stu0;
```
All members are unknown values

Structure variables
stu0, stu1, stu2

declaration/initialization
```
student stu1 = {.id=245,
                .name="john",
                .gpa=3.5};
```
Order doesn't matter

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
1	++ --	Suffix/postfix increment and decrement
	()	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                 // accesses the id member value
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

- Manipulation
  - Only one operation full structure operator – assignment

```
stu2 = stu1;          // copy all member values from stu1 to stu2
                      // must be the same structure 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 to structures
  - 2 ways to access a member value from a pointer to a structure

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
```

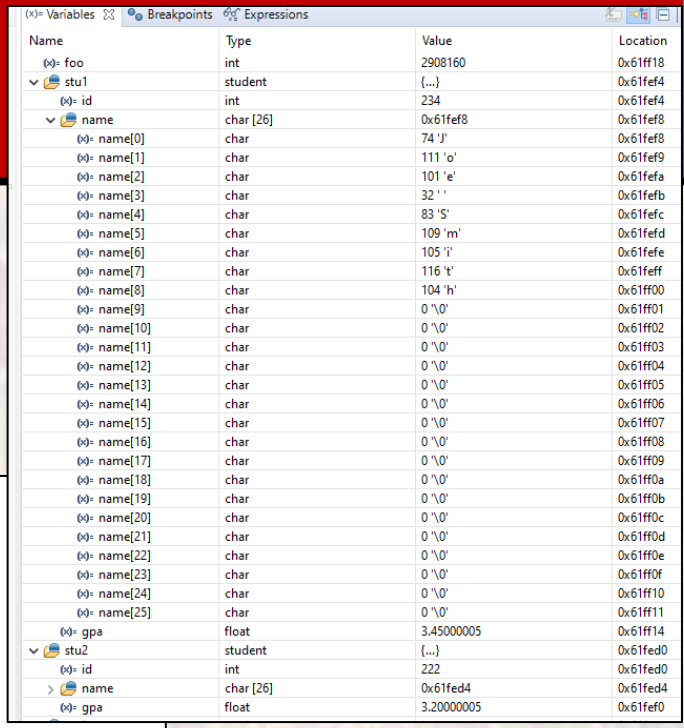
Precedence	Operator	Description
1	++ --	Suffix/postfix increment and decrement
	()	Function call
	[]	Array subscripting
	.	Structure and union member access
	->	Structure and union member access through pointer
	(*type*){*list*}	Compound literal(C99)

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

- Example




```
//////////////////////////////////////////////
/*  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

```
#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

pointer access

dereference

```
<terminated> (exit value: 0) Class_Cons_Project.exe [C/C++ Applic
234, Joe Smith, 3.450000
222, Sara Jones, 3.200000
13306988, ¼þa, 82363392669134228000000000000000.000000
13306988, ¼þa, 82363392669134228000000000000000.000000
234, Joe Smith, 3.450000
222, Sara Jones, 3.200000
345, ¼þa, 4.000000
345, ¼þa, 4.000000
foo = 222
```

Structures

- Example

```
////////////////////////////////////////////////////////////
///*  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
```

pointer to structure

```
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 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
```