Functions in C

Last updated 6/27/24

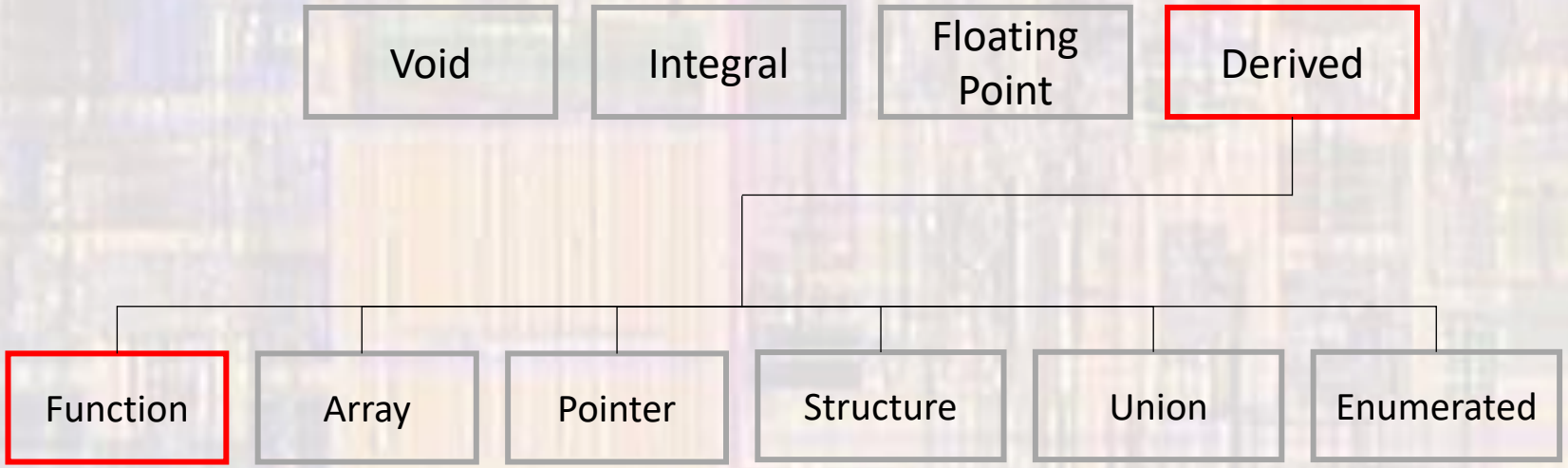
These slides introduce functions in C

Functions in C

- Purpose of Functions in Programming
  - Allow one piece of code to be reused with different inputs
  - Break problems into manageable pieces
  - Allows function libraries - to reuse common code
    - # include <stdio.h>

Functions in C

- C Types
  - Functions are a ‘type’ in C, inside the ‘derived’ group

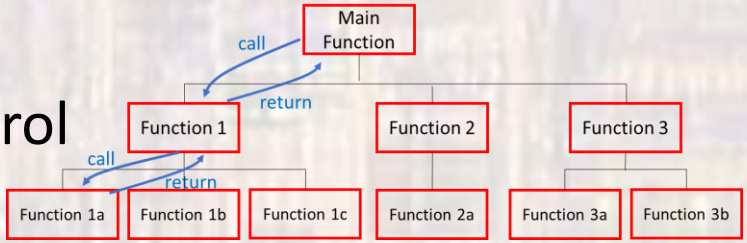


Functions in C

- C Program Structure
  - A C program is composed of a series of functions
  - main is the top level function in C
    - One and only one main function
    - main may or may not call other functions

Functions in C

- Control Chart
  - All communication must go through the Calling/Called function path
  - Calling (current) function has control
  - Calling function calls a function
  - The called function receives control
    - Now the current function
  - When done – the called function returns control to the calling function



Functions in C

- C Function – simplified view
  - Receive zero or more pieces of data (actual parameters)
    - Note: the value of the variables are passed to the function – not the variables themselves
  - Operate on the data

    and/or
  - Have a side effect
  - Return zero or one piece of data (return value)

Functions in C

- User Defined Functions
  - 3 parts to every C function

Declaration

```
// Function Declarations (prototypes)
void greeting(void);
```

declaration must come before the first use of the function
(tells the compiler what to expect)

Call

```
int main(void){
  …
  greeting(void);
  return 0;
}
```

If no data is passed to the function we can use (void) or ()

Definition

```
// Function Definition
void greeting(void){
  printf("Hello EE1910");
  return;
}
```

This function only has a side effect

Even if nothing is being returned we should include a return statement

Functions in C

- User Defined Functions – Definition
  - Defines the actions performed by the function

  - Function definition structure

```
return-type function-name(formal parameter list){
        statements;
        return return_value;
}
```

Formal parameter list structure
param-type param-name, param-type param-name, …

```
float myFunction(int x, float y, char z){
    float val;
    val = x * y - z;
    return val;
}
```

This declares variables for use inside the function
- to store the values passed to the function

Functions in C

- User Defined Functions – Call
  - Transfers control to the function along with passing parameters and accepting the return value

  - Function call structure

```
function-name(actual parameter list);
   or
var = function-name(actual parameter list);
   or
if (function-name(actual parameter list) == 0){
…
```

```
myFunction(a,b,c);

foo = myFunction(a,b,c);

if(myFunction(a,b,c) == 12){
```

The types for a,b,c and foo must match the function definition

Functions in C

- User Defined Functions – Declaration
  - Used by the compiler to determine if there are any syntax errors in the code
  - Function declaration structure

    return-type function-name(formal parameter list);

    Formal parameter list structure
    param-type param-name, param-type param-name, …

    int myFunction(int x, float y, char z);

  - Types must match function definition
  - Strongly encourage names match also
  - Just a copy of the first line of the definition with a ;

Functions in C

- Single File Program Structure

Includes

Function Declarations

```
void main(void){
  …
  foo = fun1(a, b);
  fun2(2, c);
  if(fun1(c, d)) {
    …
  }
}
```

Function 1 Definition

Function 2 Definition