Functions

Last updated 9/1/21

Functions

- In algebra we are familiar with functions

definition

$$ave(a,b) = (a + b)/2$$

call

$$foo = 2 + 7 + ave(3,4)$$
$$2 + 7 + 3.5 \longleftarrow \quad result$$

- a and b are called "formal parameters" in the definition
- 3 and 4 are called "actual parameters" in the function call
- The result of the function is called the "result"

Functions

- In algebra we are familiar with functions

definition   ave(a,b) = (a + b)/2

x = 5   x and y are replaced with
                                                   5 and 7 when we evaluate
y = 7   the function

call   foo = 2 + 7 + ave(x,y)

        2 + 7 + 6

- a and b are called "formal parameters" in the definition
- x and y are called "actual parameters" in the function call
- The result of the function is called the "result"

Functions

- Why Functions?

  - 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>
    - #include <math.h>
    - #include "mbed.h"

Types

- Functions are a 'type' in C, inside the 'derived' group

5
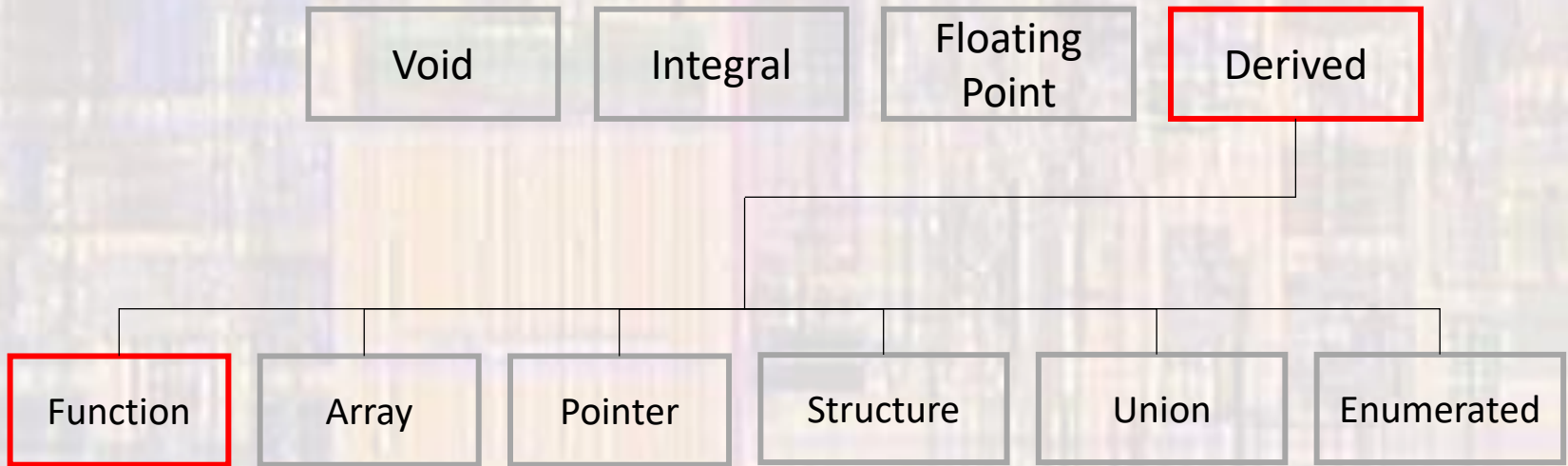
Functions

- 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 may or may not call other functions, …

Program Structure

- ## Control Flow - functions
  - All communication must go through the Calling/Called function path
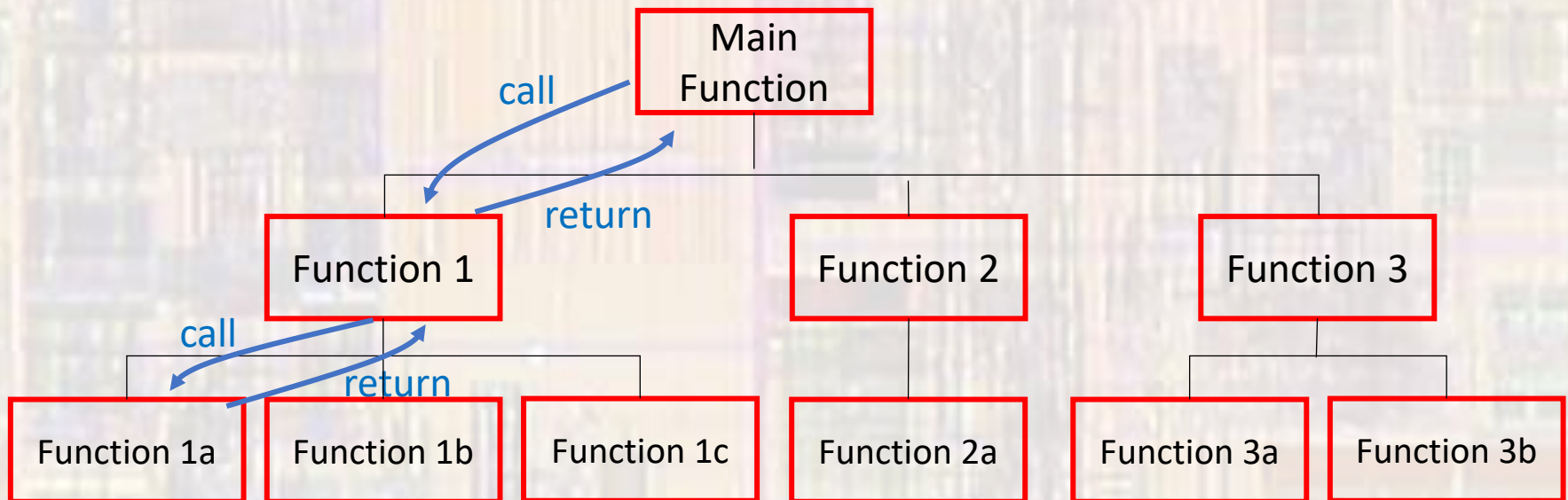
Program Structure

- Control Flow - functions
  - All communication must go through the Calling/Called function path

  - Calling function has control

  - Calling function calls a function

  - The called function receives control

  - When done – the called function returns control to the calling function (return)

Functions

- Function – simplified view

  - Receive zero or more pieces of data (parameters)
  - Operate on the data
  - Potentially have a side effect
  - Return zero or one piece of data (return value)

Functions

- User Defined Functions
  - Order of appearance
    - Declaration
    - Call
    - Definition

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

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

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

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

This function only has a
side effect

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

Even if nothing is being
returned we should include
a return

Functions

- User Defined Functions - Definition

  - 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;

    }

    > The formal parameter list serves to declare local (to the function) variables. These local variables are then assigned the values passed as actual parameters in the function call.
    > When the function returns – all local (to the function) variables are no longer accessible

Functions

- User Defined Functions - Call

  - 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){

To be clear – the variables (if any) in the actual parameter list are not passed to the function – the values of the variables in the actual parameter list are passed to the function

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

Functions

- User Defined Functions - Declaration

  - 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

- User Defined Functions - example

declaration
```
float vol(float length, float width, float height);
…
```

```
int main(void){
  float volume;
  float W;
  float L;
  float H;
  // enter W, L, H
  …
```

call
```
  volume = vol(L, W, H);
  …
  return 0;
}
```

definition
```
float vol(float length, float width, float height){
  float tmp_val;
  tmp_val = length * width * height;
  return tmp_val;
}
```

Actual Parameters

Formal Parameters

W=5
L=3
H=2

Values passed

volume = vol(3,5,2);

volume = 30

length = 3
width = 5
height = 2

Values assigned

return 30

Functions

- User Defined Functions - example

declaration

```
float ave(float val1, float val2);
…

int main(void){
  float average;
  float try1;
  float try2;

  // enter try1, try2
  …
```
call
```
  average = ave(try1, try2);
  …
  return 0;
}
```

definition
```
float ave(float val1, float val2){
  float tmp_val;
  tmp_val = (val1 + val2)/2;
  return tmp_val;
}
```

Actual Parameters

Formal Parameters

try1=5.5
try2=3.3

average = ave(5.5, 3.3);

average = 4.4

val1 = 5.5
val2 = 3.3

return 4.4

Functions

- Functions and Memory – program memory

  - Function call transfers execution to a separate section of code (function)

  - When done, the function returns to the next line of code

  - Multiple calls to the same function transfer execution to the same location
    - One copy of the function
    - Data space is reused (typically)

Program Memory

Exception Vectors
Executing instructions in line
Function Call
Text Executable Code
Function Code Return
Read Only - Data
Copy of RAM Data section
Boot Loader
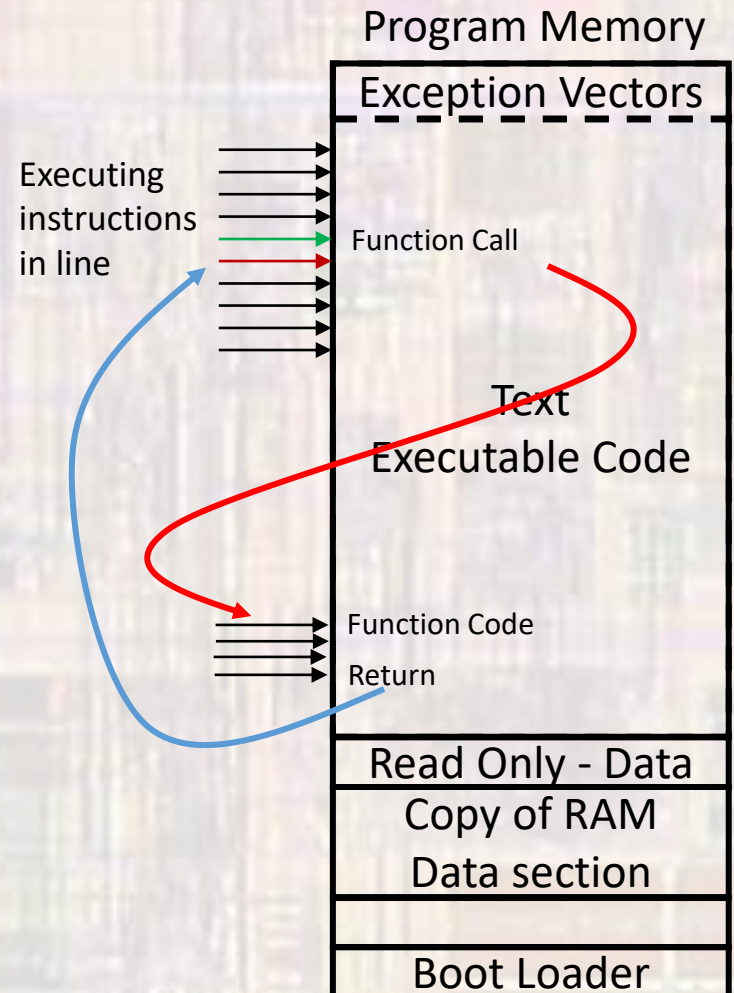
Functions

- Functions and Memory – data memory

  - Function call creates a space in the stack
    - copy of the actual parameters
    - any function variables (local variables)

  - Function operates in this newly created space (scope)

  - When the function returns, the space is reclaimed (not necessarily erased but no longer available)

Data Memory

Stack
Function memory Copy of parameters Local variables
↓
↑
Heap
Bss
Data
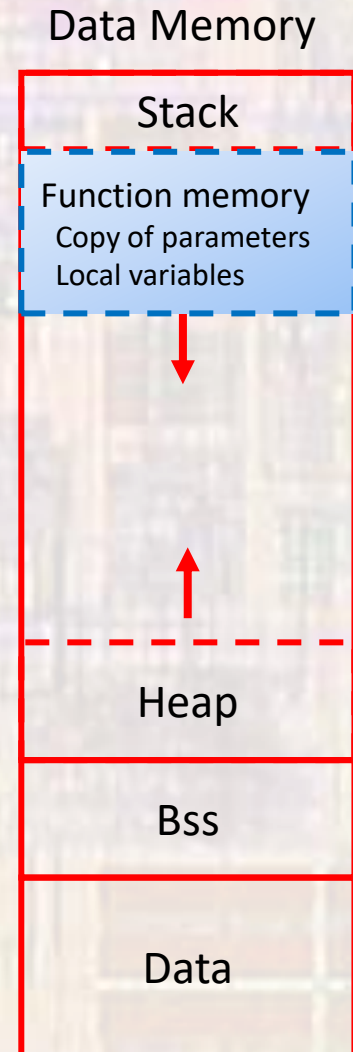
Functions

- ## User Defined Functions – Data memory

Data Memory

declaration

```
float ave(float val1, float val2);
…

int main(void){
    float average;
    float try1;
    float try2;

    // enter try1, try2
    …
```

call

```
    average = ave(try1, try2);
    …
    return 0;
}
```
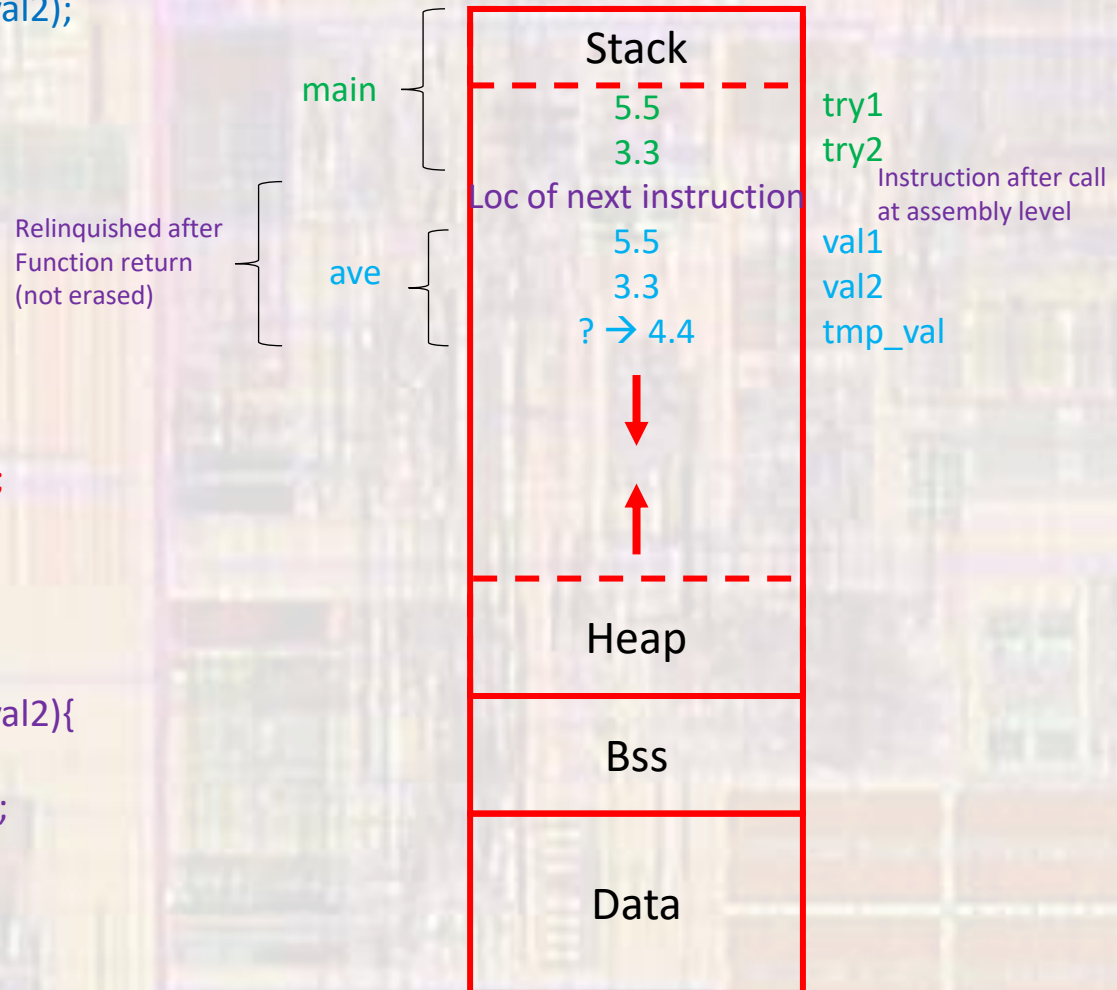
definition

```
float ave(float val1, float val2){
    float tmp_val;
    tmp_val = (val1 + val2)/2;
    return tmp_val;
}
```

Stack	
5.5	try1
3.3	try2
Loc of next instruction	Instruction after call at assembly level
5.5	val1
3.3	val2
? → 4.4	tmp_val

main

ave

Relinquished after Function return (not erased)

Heap

Bss

Data

Functions

- Functions and Memory

```
void update_acct(float balance, float int_rate);

int main(void){
    float balance;
    float int_rate;
    …
    printf("balance is: %f", balance);
    update_acct(balance, int_rate);
    printf("balance is: %f", balance);
    return 0;
}

void update_acct(float balance, float int_rate){
    balance = balance + balance * int_rate;
    printf("balance is: %f", balance);
    return;
}
```

with the original balance of 10,000 and a 10% int_rate – what does this print

Functions

- Functions and Memory

```
int main(void){
  float balance;
  float int_rate;

   …
  printf("balance is: %.2f\n", balance);
  update_acct(balance, int_rate);
  printf("balance is: %.2f\n", balance);
  return 0;
}

void update_acct(float balance, float int_rate){
  balance = balance + balance * int_rate;
  printf("balance is: %f\n", balance);
  return;
}
```

balance is 10000.00
balance is 11000.00
balance is 10000.00

Variables are not passed to the function – values are

Changes in the function are made to the local copies of the variables

Functions

- ## Functions and Memory

```
void update_acct(float balance, float int_rate);

int main(void){
    float bal;
    float ir;
     …
    printf("balance is: %f", bal);
    update_acct(bal, ir);
    printf("balance is: %f", bal);
    return 0;
}

void update_acct(float balance, float int_rate){
    balance = balance + balance * int_rate;
    printf("balance is: %f", balance);
    return;
}
```

Choosing names carefully can prevent some confusion