Functions

Last updated 12/14/20

Functions

- These slides introduce the functions in C
- Upon completion: You should be able interpret and code using functions

Functions

- In algebra we are familiar with functions

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

call $foo = 2 + 7 + ave(3,4)$

$2 + 7 + 3.5$ ← result

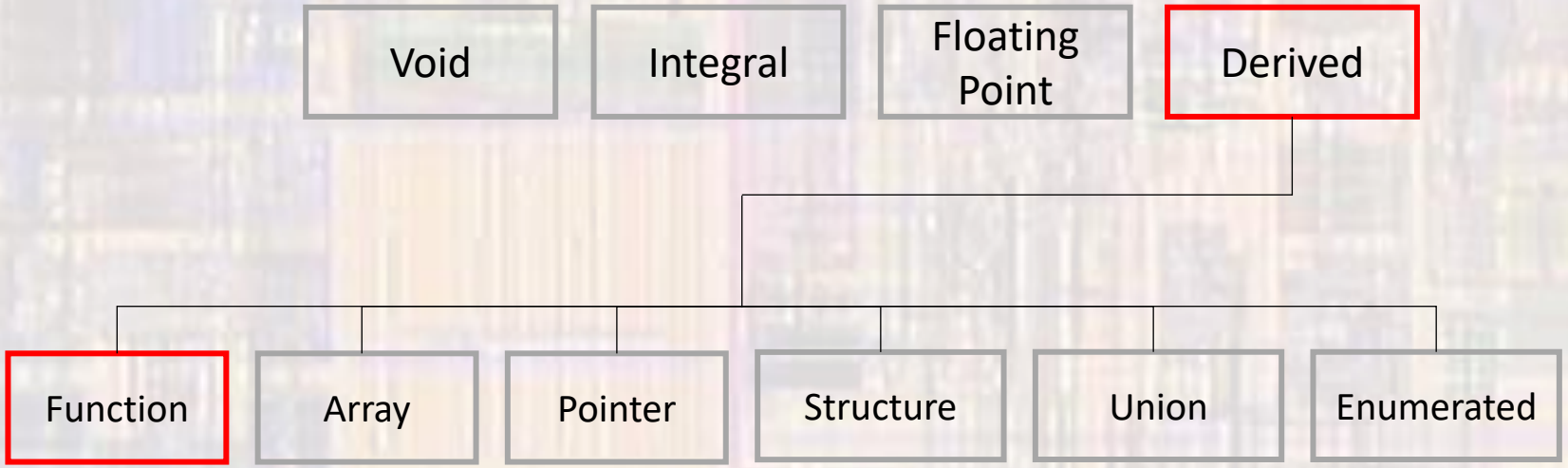
- a and b are called "formal parameters" in the definition
- 3 and 4 are called "actual parameters" in the function call
- The result of the function is called the "result"

Functions

- Function Purpose
  - 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>

Types

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

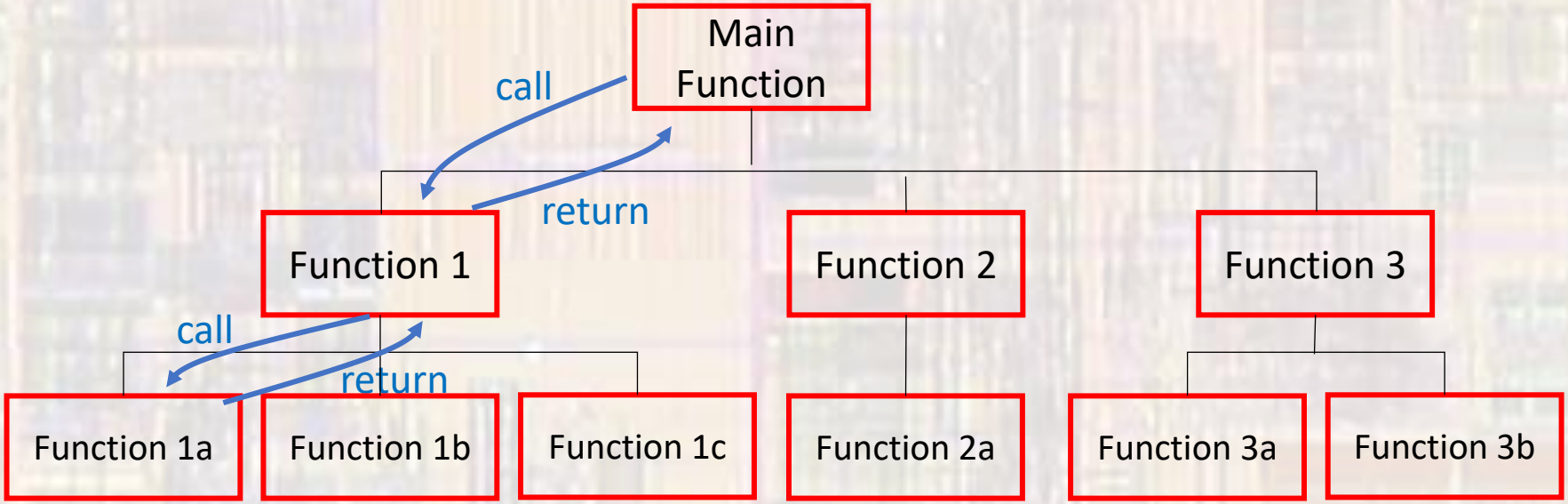


Functions

- 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

Program Structure

- Structure Chart
  - Allows solutions to be broken into manageable, understandable and logical pieces
  - All communication must go through the Calling/Called function path



Program Structure

- Structure Chart
  - 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

Functions

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

Functions

- User Defined Functions
  - Declaration
  - Call
  - Definition

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


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


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

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

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

This function only has a side effect

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;
}
```

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){
```

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

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

call

definition

Actual Parameters

Formal Parameters

W=5
L=3
H=2

volume = vol(3,5,2);

volume = 30

length = 3
width = 5
height = 2

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

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

call

definition

Actual Parameters

Formal Parameters

try1=5.5
try2=3.3

average = ave(5.5, 3.3);

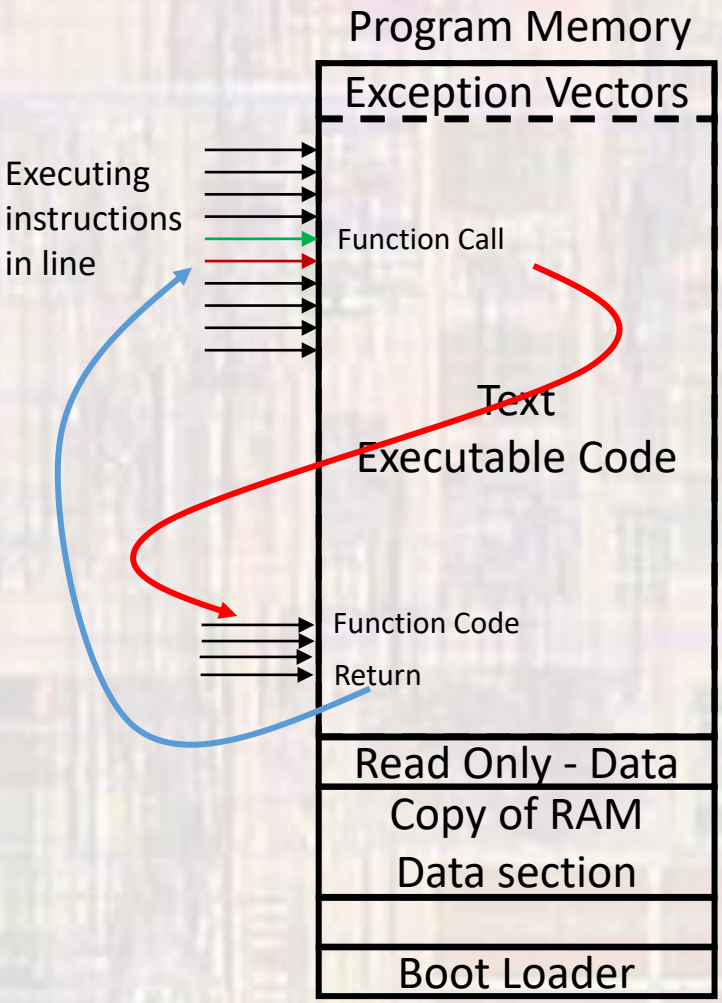
average = 4.4

val1 = 5.5
val2 = 3.3

return 4.4

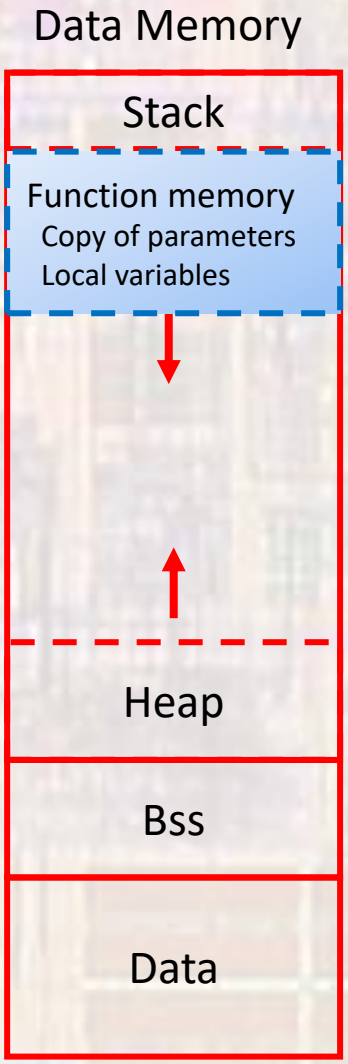
Functions

- 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)



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)



Functions

- User Defined Functions – Data memory

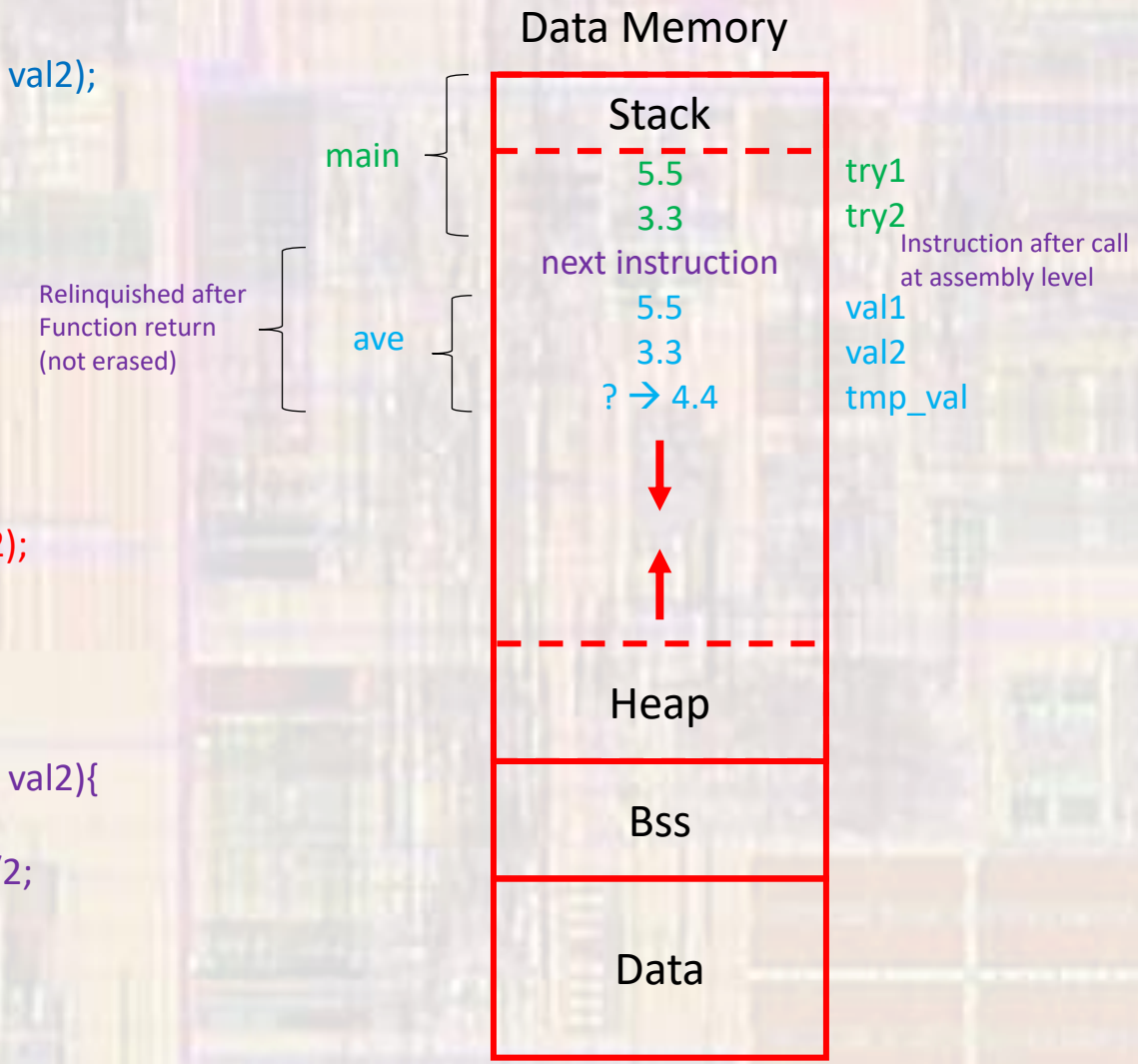
declaration

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

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

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

call

definition

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



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

Changes in the function are made to the local copies of the variables – not to the copy in main

The local copies are reclaimed when the function returns

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