Functions

Last updated 6/16/23

These slides introduce functions in C

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

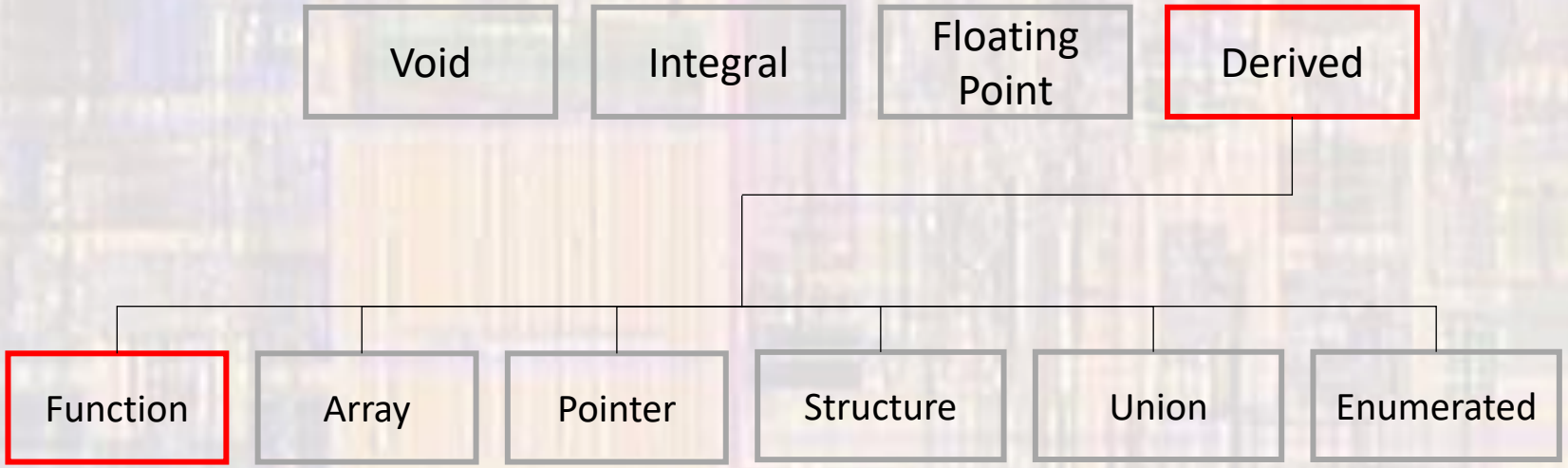
- a and b are called "formal parameters" in the definition
- 3 and 4 are called "actual parameters" in the function call
- The result of the function is called the "result"

Functions

- 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

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

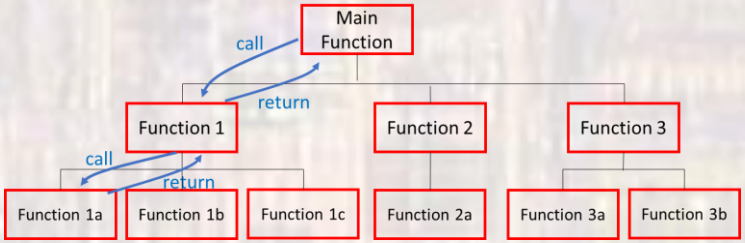


Functions

- 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

- Control 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)
    - 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

- 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

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 statement

Functions

- 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

- 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

- 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

- User Defined Functions – example 1

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;
}
```

call

Actual Parameters

Formal Parameters

definition

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

values passed - not the variables

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 2

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

values passed - not the variables

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

- User Defined Functions – example - stack

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

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

t0 →

  // enter try1, try2 (9, 3)
t1 →
  …
t4 →  average = ave(try1, try2);
  …
  return 0;
}

float ave(float val1, float val2){
t2 →  float tmp;
t3 →  tmp = (val1 + val2)/2;
  return tmp;
}
```

Data Memory - Stack

	t0	t1	t2	t3	t4
ave	?	?	?	?	6
try1	?	9	9	9	9
try2	?	3	3	3	3
val1			9	9	6
val2			3	3	9
tmp			?	6	3

This is a simplified representation of the stack – see the notes on non-linear function execution for a more complete picture