

# CSE 1320 - Intermediate Programming

## Functions

Alex Dillhoff

University of Texas at Arlington

# Functions

**Functions** in programming are named blocks of code that execute some number of statements.

- ▶ They have identifiers.
- ▶ They can accept arguments.
- ▶ They can return values.
- ▶ They enable **modularization** in code.

# Functions Definitions in C

Functions are **defined** in C with the following syntax:

## Syntax

```
type function_name(params) {  
    statements  
}
```

## Example

```
int is_odd(int number) {  
    return number % 2;  
}
```

# Function Types

The type of a function is defined by what it returns.

- ▶ Scalar type: `int`, `float`, etc.
- ▶ Pointers
- ▶ Structures
- ▶ Unions

# Function Identifiers

Rules for function identifiers are the same as for variable identifiers.

- ▶ Consist of numbers, letters, and underscores.
- ▶ **CANNOT** start with a number.
- ▶ **CANNOT** be a reserved word.

# Function Parameters

Functions can accept parameters or `void`.

Any parameters that are within the function header are called **formal parameters**.

The variables passed in the function call are called **actual parameters**.

# Uses of Functions

They are very useful in separating distinct tasks or chunks of logic.

Functions should be defined to complete a specific task.

Functions provide a way to communicate data between *modules* in a program.

# Intra-Program Communication

Functions communicate with other functions through the return value and their formal arguments.

- ▶ Arguments can be as general or specific as the task requires.
- ▶ Return values can be the results of a search, a program state, result of a computation, etc.



# Returning Values

Functions are not required to return anything at all. Both of the following are valid definitions:

## No Return

```
void func() {  
    return;  
}
```

## Return Value

```
float func(float a, float b) {  
    return a * b;  
}
```

# Scope in Functions

- ▶ Functions are **global** by default.
- ▶ Qualifying a function with `static` restricts their access to the file in which they are defined.
- ▶ Scalar value arguments passed into the function are **local**.
- ▶ Variables created in a function are **local**.

# Scope in Functions

**EXAMPLE:** `function_scope.c`

# Declaring a Function

Function prototypes can be declared before they are defined.

## Syntax

```
type func_name(params);
```

## Example

```
int is_odd(int);
```

**Note:** Formal parameter identifiers are not required in function declarations.

# Declaring a Function

Function declarations allow the compiler to

- ▶ Check the argument types
- ▶ Check the return type

# Using `extern`

You can include a function defined in another file using `extern`.

# Using `extern`

**EXAMPLE:** `extern_func.c`

# What happens when a function is called?

When a function is called in C, an execution environment is created.

- ▶ Memory is assigned on the stack for local variables.
- ▶ Parameters passed by value are also given memory on the stack. The original values are not modified in the function.
- ▶ The function identifier itself has an address that can be used as a parameter (more on this when we get to pointers).



# What happens when a function is called?

When a function returns, control is sent back to the calling environment.

- ▶ When in main, the OS is the calling environment.
- ▶ When in a sub-function, the control returns to the calling function.

# Function Examples

`is_prime.c stats.c`