# Functions

Darin Brezeale

The University of Texas at Arlington

# Functions

Recall the program `datecheck.c`, which compared two dates to see which comes first. What would it be like if we wanted to compare multiple pairs of dates?

See `datecheck-nofunction.c`.

You are probably thinking there should be an easier way, and you are right. For this situation, we should use a function.

# Functions cont.

Functions are collections of computer code that perform a task.

Why use functions?

- Makes general purpose of program easier to follow
- Reduces number of places to update
- Makes reuse of code easier
- Hide implementation, letting programmer focus on functionality

See `datecheck-function.c`.

# Functions cont.

When writing our own functions, the general form is

```
return_type function_name(input_type variable_name)
{
    /* something happens here */
}
```

Example:

```
int addnumbers(int number1, int number2)
{
    int sum = number1 + number2;
    return sum;
}
```

# Functions cont.

- The variable names in the function definition do not need to match the names in the function call, but the quantity should match.
- To return a value, we use the `return` keyword.
- We can declare variables in our function just as we did in `main`.
- We can call other functions from within our function.

# Return and Input Types

The types of variables that we can pass or receive from a function can be any of the types that we declare variables to be–`int`, `float`, array (actually, we pass the address of the array), etc.

What type do we use if we are not passing or not returning anything? `void`

Example:

```
void print2numbers(int number1, int number2)
{
    printf("%d + %d is %d\n", number1, number2,
           number1 + number2);
}
```

See `example-function2.c` for more examples.

# Function Declarations

We must let the compiler know about the function prior to using it by either:

- Placing the function code before `main`
- Placing a function declaration (or prototype) before `main`

# Function Declarations cont.

Example of function declaration:

```
#include <stdio.h>

/*      function declarations */
void squarenum(int);

int main(void)
{
    int x = 15;
    squarenum(x);
}

/*      function definition   */
void squarenum(int y)
{
    printf("%d squared is %d\n", y, y*y);
}
```

# Including functions

To use functions in external files, we need to tell the compiler where to find the function declarations.

Example: To use functions in the Standard C Library `stdio.h`, we place the following at the top of our program:

```
#include <stdio.h>
```

# Including functions cont.

We could place our own functions in their own file (e.g., `myfunctions.c`) and use them in our programs. Just as was the case when using functions from the Standard Library, such as `printf()`, we need to include information at the top of our program letting the compiler know where it can find the function declarations, for example

```
#include "myfunctions.h"
```

We will do this later in the course.

# Variable Scope

We need to know the following when using variables in functions:

- The process used in this lecture for providing variable values to our function is called *pass by value*. When doing so, a copy of the variable is provided.
- Variables declared outside the function are unknown to the function unless we pass them.
- Variables declared within a function block are known only to that function.

See `example-function-scope.c` on course webpage.