COP 3223H: Introduction to C Programming

Fall 2023

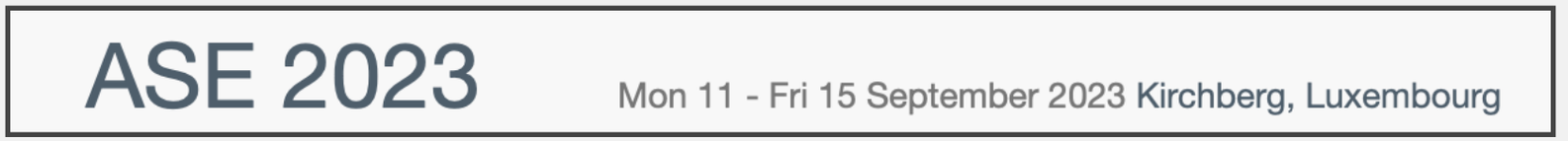
University of Central Florida

Dr. Kevin Moran

Week 3- Class 1: User Defined Functions I

Administrivia

- *Small Programming Assignment One*
  - Due on Monday (Sept 11th)
  - Test script coming soon
- *Quiz 1 will be posted Friday*
  - Due on September 15th (timed, one try!)
- *Upcoming Recorded/Virtual Lectures*
  - Dr. Moran will be traveling next week for ASE'23 in Luxembourg
  - *Friday Sept 8th - Weds Sept 13th* will be ***recorded lectures*** (Videos posted to Webcourses)
  - *Friday Sept 15th* will be a *Virtual lecture (Zoom)*



Today's Agenda

1. Review the Quadratic Equation Program
2. Begin Introduction of User-defined Functions

Review and Quadratic Equation Program

Recap of Last Class - Arithmetic Exp.

- Lots of problems that programmers solve involve the use of formulating mathematical expressions.
- In this course we focus on arithmetic operations (think algebra level)
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Modular (modulus)

Quadratic Equation Program

Recap of Last Class (cont.)

- Pre-defined C functions from various libraries
- Everything we have done has always been implemented in the main function
- If we want to do a lot of computation, the main function will grow large and unwieldy

What if we could organize our code in a way that separates a set of instructions that complete semantically coherent tasks?

Introducing Functions!

- All C programs execute instructions in functions.
- All the programs we've looked at only had 1 function (`main`).
- Programs can get bigger based on the problem trying to be solved.
- It would be very convoluted with all instructions executed in the main function (*this is bad practice!*).
- In this lecture, we will learn that programmers can define their own user-defined functions to perform tasks.

Benefits of Defining our Own Functions

- Improves Readability
- Improves Reusability
- Helps to organize the abstractions in your program

User-defined Functions

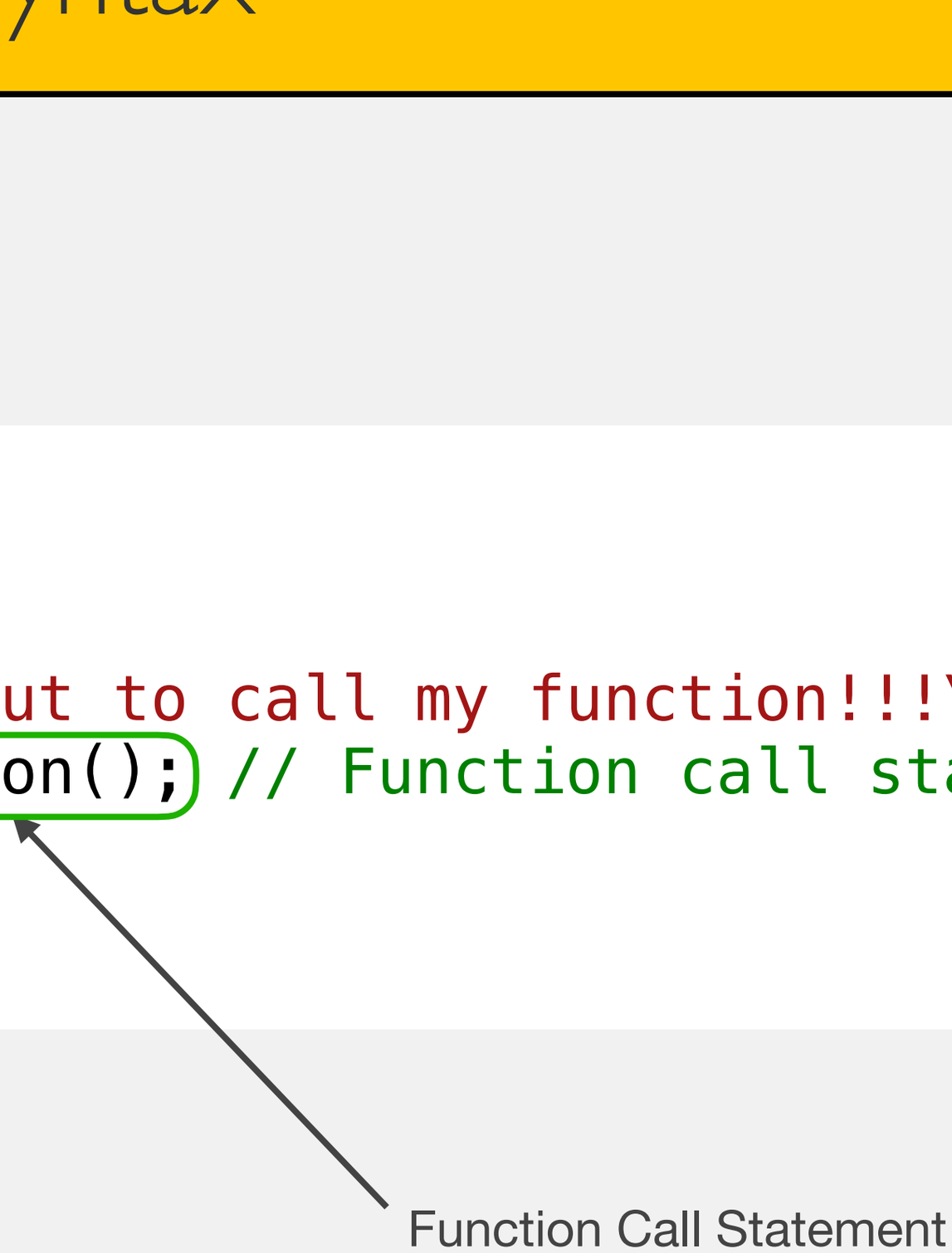
- One way that programmers implement top-down design is defining their own functions (user defined functions)
- User defined functions are sets of instructions that are ***defined*** by the programmer
- Programmers will break down a larger problem into subproblems and will solve these subproblems in user-defined functions
- In order to invoke the function, you must ***call*** it

Function Call Syntax

```c
int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

}
```



Function Prototypes

- Just like variables, functions must be declared as well.
- Prototypes allows the Operating System know how much memory space needs to be reserved based on the return type and arguments.

Function Prototype

```c
#include <stdio.h>

void myOwnFunction();

int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Function Definitions

- Just like declaring a variable, you must assign it a value.
- For function definitions, you must write out the set of instructions to perform the task that needs to be written out.

Function Definition

```c
#include <stdio.h>

void myOwnFunction();

int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Tracing C Program with User Defined Function

Here

```c
#include <stdio.h>

void myOwnFunction();

int main() {
    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Tracing C Program with User Defined Function

Here

```c
#include <stdio.h>

void myOwnFunction();

int main() {
    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Tracing C Program with User Defined Function

Here

```c
#include <stdio.h>

void myOwnFunction();

int main() {
    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Tracing C Program with User Defined Function

Hold

Here

```c
#include <stdio.h>

void myOwnFunction();

int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Tracing C Program with User Defined Function

```c
#include <stdio.h>

void myOwnFunction();

int main() {
    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Hold

Here

Tracing C Program with User Defined Function

```c
#include <stdio.h>

void myOwnFunction();

int main() {
    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Hold

Here

Tracing C Program with User Defined Function

```c
#include <stdio.h>

void myOwnFunction();

int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Here

Tracing C Program with User Defined Function

```c
#include <stdio.h>

void myOwnFunction();

int main() {

    printf("About to call my function!!!\n");
    myOwnFunction(); // Function call statement

    return 0;

}

void myOwnFunction()
{
    printf("This is my awesome function!!!\n");
}
```

Here

Types of Functions

- There are ***<u>two types</u>*** of functions.
  - Functions that *return a value*.
  - Functions that *don't return a value*.
- These types of functions are defined through their prototypes.
  - Functions that don't return a value have the reserved word **`void`** in front of the name.
  - Functions that do return a value have the type of data (**`int`**, **`double`**, **`char`**) in front of the of the name.

Example of Function that Does Not Return Value

```c
#include <stdio.h>

void myFavoriteNumber();

int main()
{
    myFavoriteNumber();

    return 0;
}

void myFavoriteNumber()
{
    int num;

    printf("Enter your favorite number: ");
    scanf("%d", &num);

    printf("Your favorite number is %d\n", num);
}
```

Example of Function that Does Return Value

```c
#include <stdio.h>

int myFavoriteWholeNumber();

int main()
{
    myFavoriteNumber();
    return 0;
}

int myFavoriteWholeNumber()
{
    int num;

    printf("Enter your favorite number: ");
    scanf("%d", &num);

    printf("Your favorite number is %d\n", num);
}
```

Acknowledgements

Slides adapted from Dr. Andrew Steinberg's COP 3223H course