# CSE 142
# Computer Programming I

## Function Parameters





## Overview

Many concepts in this lecture! The most memorable:

- Function parameters and arguments
- Return values, return types, and the return statement
- Local variables
- Function prototypes and header files

Lots of new terminolgy, too

Near the end of the lecture:

- An extended program, traced



## Refresher: Printing a Banner

Our original banner program called this function to print a very simple banner

```
/* write separator line on output */
void PrintBannerLines (void)
{
    printf("**********\n");
    printf("**********\n");
}
```



## The Client Wants a Change

Suppose we now want to change the program: it should now print 5 rows of asterisks when it starts and when it finishes, but print the original 2 line banner everywhere else

We could write an additional function that prints 5 rows of asterisks, or...



## Can we Generalize?

Suppose we now want to change the program: it should now print 5 rows of asterisks when it starts and when it finishes, but print the original 2 line banner everywhere else

We could write an additional function that prints 5 rows of asterisks, or...

Could we somehow generalize PrintBannerLines? Could we make the same function do double duty?



## Can we Generalize?

Can we modify the function so that instead of

***print two rows of asterisks***

it will:

***print N rows of asterisks***

where N is the number of rows that we want "this time" when we call it

N is information that the function needs to know

```
#include <stdio.h>
int main(void)
{
   PrintBannerLines(5);

   /* produce some output */

   PrintBannerLines(2);

   /* produce final output */

   PrintBannerLines(5);

   return 0;
}
```

5

Code for PrintBannerLines

The value in the parentheses is called the "parameter" of this call.

## Code for the Modified Function

The function will start off this way:

```
void PrintBannerLines (int n)
{
   ...
```

n is the "parameter" of the function. n can be used inside the function just like a variable

The full solution won't be shown now. It requires a feature called "iteration" that we will cover later. We'll see parameters in other examples.

G2-8

## A New Example Problem

Specification: Write a function which, given the radius, computes and returns the area of a circle with that radius

The new wrinkle here is that the function must "return" a value

G2-9

## Returned Values

Parameters are a way for the calling routine to "send data" to the function

The new concept, return values, are the opposite, a way for the function to send data back to the calling routine

G2-10

## area Function, Solved

Specification: Write a function which, given the radius, returns the area of a circle with that radius

New features:

1. The return statement stops function execution and specifies the value returned.
2. The type of the returned value is stated before the function name

```
/* return area of circle with radius r */
double area (double r)
{
   return  3.14 * r * r;
}
```

G2-11

## Void Parameters, Non-void Returns

This function returns a number that it generates internally, without the need for a parameter (information) supplied by the caller.

```
/* return a "random" number. */
double GenRandom (void)
{
   double result;
   result = ...
   return result;
}
```

function type (type of returned value). We say "*GenRandom( )* is a function of type *double*" or "*GenRandom( )* returns a *double*."

*local variable* – exists only while function is executing

returned value

*return* statement

G2-12

## More on *return*

For *void* functions:
- return; causes control flow to return to the statement following the call in the caller.

For functions that return a value:
- return *expression;* causes control flow to return to the caller. The function call is "replaced" with the returned value.

*Note: no parentheses are needed around the expression*
- return is a C statement. It is not a function call

G2-13

## Calling a Non-Void Function

A value-returning function can be used anywhere an expression of the same type can be used

```
int main (void) {
   double   firstRandom, secondRandom;
   double   result;
   firstRandom =  GenRandom( );
   secondRandom =  GenRandom( );
   result = firstRandom + secondRandom;
   printf("the value of %f + %f is %f.",
          firstRandom, secondRandom, result);
   return 0;
}
```

G2-14

## *return* in *void* Functions

optional

```
/* print banner line */
void print_banner (void)
{
     printf("**************\n");
     printf("**************\n");
     return;
}
```

```
/* do something */
void example (void)
{
     int no_reason_to_continue;
     ...
     if (no_reason_to_continue) {
          return;
     }
     ...
}
```

terminate function execution before reaching the end (required)

G2-15

## Discussion Questions

1. Can you have more than one *return* inside a function?
2. Does a *return* statement have to be the last statement of a function?
3. If a function starts off as
   *double calculation (void) {…*
   could it contain this statement?
   *return;*
4. If a function starts off as
   *void printfBankBalance (void) {…*
   could it contain this statement?
   *return currentBalance;*

G2-16

## Matching up the Arguments

Rule: The function call must include a matching argument for each parameter.

When the function is executed, the value of the argument becomes the initial value of the parameter.

```
int main (void)
{  ...
   z = 98.76;
   x = 34.575 * area ( z/2.0 );
   ...
   return 0;
}
```

```
/* Find area of circle with radius r */
double area (double r)
{
   return   3.14 * r * r;
}
```

## More Terminology Confusion

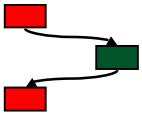
Many people use the term *formal parameter* instead of *parameter* and *actual parameter* instead of *argument*. We will try to stick to *parameter* and *argument* for simplicity, but the other terminology will probably slip in from time to time.

People often refer to replacing a parameter with the argument in a function call as "passing the argument to the function".

G2-18

## Review: Function Control Flow

Some time ago we described the basic flow.



We can now give a much more detailed account of how this flow works

G2-19

## Control and Data Flow

When a function is called:

1. Memory space is allocated for the function's parameters and local variables
2. Argument values are copied;
3. Control transfers to the function body;
4. The function executes;
5. Control and return value return to the point of call.

G2-20

## Control and Data Flow

```
int main (void)
{
   double x, y, z;
   y = 6.0;
   x = area(y/3.0) ;
   ....
   ....
   z = 3.4 * area(7.88) ;
   ....
  return 0;
}
```

```
/* Find area of circle
   with radius r */
double area (double r)
{
   return  3.14 * r * r;
}
```

2.0

12.56

7.88

194.976...

G2-21

## Control and Data Flow (2)

```
int main (void)
{
   double x, y, z;
   y = 6.0;
   x = area(y/3.0) ;
   ....
   ....
   z = 3.4 * area(7.88) ;
   ....
  return 0;
}
```

2.0

12.56

7.88

194.976...

G2-22

## Style Points

The comment above a function must give a complete specification of what the function does, including the significance of all parameters and any returned value.

Someone wishing to use the function should be able to cover the function body and find everything they need to know in the function heading and comment.

```
/* Compute area of circle with radius r */
double area (double r)
```

G2-23

## Multiple Parameters

A function may have more than one parameter

Arguments must match parameters in number, order, and type

```
double gpt, gpa;
gpt = 3.0 + 3.3 + 3.9;
gpa = avg ( gpt, 3 );
...
```

```
double avg (double total, int count)
{
  return  total  /  (double) count ;
}
```

arguments

parameters

G2-24

## Rules for Returns

A function can only return one value --but it might contain more than one *return* statement

In a function with return type T, the returned expression must be of type T.

A function with return type T can be used anywhere an expression of type T can be used.

G2-25

## Where Are We?

We have seen all of the basic concepts for how a function communicates with the outside world, through parameters and return values

We know the syntax involved, as well as the logical concepts

There is still a topic centered with the internal programming of the function: the use of local variables

G2-26

## Local Variables

A function can define its own local variables.

The locals have meaning only within the function.

- Local variables are created when the function is called.
- Local variables cease to exist when the function returns.

Parameters are also local.

G2-27

## A Function with Local Variables

```
/* return area of circle with
   radius r */

double CircleArea (double r)
{
   double rsquared, area;
   rsquared = r * r ;
   area = 3.14 * rsquared ;
   return   area;
}
```

parameter

local variables

G2-28

## Global Variables

C lets you define variables that are not inside any function.

- Called "global variables."

Global variables have legitimate uses, but for beginners, they often are:

- a crutch to avoid using parameters
- poor style

G2-29

## Surgeon General's Warning

In this course:

global variables are completely verboten! Only *local* variables are allowed in homework programs

Note: *#define* symbols are global, but technically, they are not variables

Their use is encouraged!

G2-30

## Local Variables: Summary

(Formal) parameters and variables declared in a function are local to it:

- cannot be accessed (used) by other functions except by being passed as actual parameters or return values)

Allocated (created) on function entry, de-allocated (destroyed) on function return.

(Formal) parameters initialized by copying value of argument (actual parameter). ("Call-by-value")

A good idea? YES!

- localize information; reduce interactions.

G2-31

## Now We're Ready!

Once we have local variables, we can develop an extended and realistic example of function usage.

Problem: Find the area of a washer-shaped figure.

Within the solution, the circleArea function already programming will be used.

P.S. The best way to follow this part of the lecture would be to have a printed copy of the full program in front of you

G2-32

## Washer Area Function

```
/* Find area of washer with given
   inner and outer radius.  */
double WasherArea (double inner, double outer)
{
    double innerArea, outerArea, areaOfWasher ;

    innerArea = CircleArea (inner) ;
    outerArea = CircleArea (outer) ;
    areaOfWasher = outerArea - innerArea;
    return areaOfWasher ;
}
```

G2-33

## The Full Program on One Page

```
#include <stdio.h>
#define PI 3.0
/* yield area of circle with radius r */
double CircleArea(double r) {
double y, area;

    y = r * r ;
    area = PI * y ;
    return  area;
}
/* yield area of a washer with ... */
double WasherArea(double inner,
double outer) {
    double innerArea, outerArea,
    areaOfWasher;
    innerArea = CircleArea(inner) ;
    outerArea = CircleArea(outer) ;
    areaOfWasher = outerArea -
    innerArea ;
    return  areaOfWasher ;
}
```

```
/* read washer info and print area */
int main(void)
{
    double inner, outer, y ;

    printf ("Input inner radius and
            outer diameter: ") ;
    scanf (" %lf %lf ", &inner, &outer) ;
    y = WasherArea (inner, outer/2.0) ;

    printf (" %f ", y) ;

    return 0 ;
}
```

G2-34

## Full Program, Page I of 2

```
#include <stdio.h>
#define PI 3.0

/* Find area of circle with
   radius r */
double CircleArea(double r)
{
    double y, area;

    y = r * r ;
    area = PI * y ;
    return  area;
}
```

```
/* Find area of a washer with
   given inner and outer area */
double WasherArea(double
                  inner, double outer)
{
    double innerArea, outerArea,
           areaOfWasher;
    innerArea = CircleArea(inner) ;
    outerArea = CircleArea(outer) ;
    areaOfWasher = outerArea -
    innerArea ;
    return  areaOfWasher ;
}
```

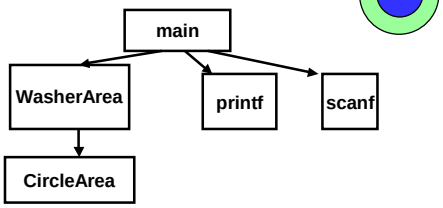
G2-35

## Full Program, Page 2 of 2

```
/* read washer info and print area */
int main(void)
{
    double inner, outer, y ;

    printf ("Input inner radius and outer diameter: ") ;
    scanf (" %lf %lf ", &inner, &outer) ;
    y = WasherArea (inner, outer/2.0) ;

    printf (" %f ", y) ;

    return 0 ;
}
```

G2-36

## Showing How Functions are Related




This "static call graph" shows who calls who



## Local Variables of main

| main | | |
|---|---|---|
| inner | outer | y |
| | | |



## Parameters and local variables of WasherArea

| WasherArea | | | | |
|---|---|---|---|---|
| inner | outer | innerArea | outerArea | areaOfWasher |
| | | | | |



## Parameters and local variables of CircleArea

| CircleArea | | |
|---|---|---|
| r | y | area |
| | | |



## Full Program, Page 2 of 2

```
/* read washer info and print area */
int main(void)
{
    double inner, outer, y ;

    printf ("Input inner radius and outer diameter: ") ;
    scanf (" %lf %lf ", &inner, &outer) ;
    y = WasherArea (inner, outer/2.0) ;

    printf (" %f ", y) ;

    return 0 ;
}
```



## Full Program, Page I of 2

```
#include <stdio.h>
#define PI 3.0

/* Find area of circle with
   radius r */
double CircleArea(double r)
{
    double y, area;

    y = r * r ;
    area = PI * y ;
    return  area;
}
```

```
/* Find  area of a washer with
   given   inner and outer area */
double WasherArea(double
                  inner, double outer)
{
    double innerArea, outerArea,
           areaOfWasher;
    innerArea = CircleArea(inner) ;
    outerArea = CircleArea(outer) ;
    areaOfWasher = outerArea -
    innerArea ;
    return  areaOfWasher ;
}
```

## Execution Trace

| main | | |
|---|---|---|
| inner | outer | y |
| 2.0 | 10.0 | |

| CircleArea | | |
|---|---|---|
| r | y | area |
| 2.0 | 4.0 | 12.0 |

| WasherArea | | | | |
|---|---|---|---|---|
| inner | outer | innerArea | outerArea | areaOfWasher |
| 2.0 | 5.0 | | | |



## Full Program, Page I of 2

```
#include <stdio.h>
#define PI 3.0

/* Find area of circle with
   radius r */
double CircleArea(double r)
{
    double y, area;

    y = r * r ;
    area = PI * y ;
    return  area;
}
```

```
/* Find  area of a washer with
   given  inner and outer area */
double WasherArea(double
                  inner, double outer)
{
    double innerArea, outerArea,
           areaOfWasher;
    innerArea = CircleArea(inner) ;
    outerArea = CircleArea(outer) ;
    areaOfWasher = outerArea -
    innerArea ;
    return  areaOfWasher ;
}
```



## Execution

| main | | |
|---|---|---|
| inner | outer | y |
| 2.0 | 10.0 | |

| CircleArea | | |
|---|---|---|
| r | y | area |
| 5.0 | 25.0 | 75.0 |

| WasherArea | | | | |
|---|---|---|---|---|
| inner | outer | innerArea | outerArea | areaOfWasher |
| 2.0 | 5.0 | 12.0 | | |



## Full Program, Page I of 2

```
#include <stdio.h>
#define PI 3.0

/* Find area of circle with
   radius r */
double CircleArea(double r)
{
    double y, area;

    y = r * r ;
    area = PI * y ;
    return  area;
}
```

```
/* Find  area of a washer with
   given  inner and outer area */
double WasherArea(double
                  inner, double outer)
{
    double innerArea, outerArea,
           areaOfWasher;
    innerArea = CircleArea(inner) ;
    outerArea = CircleArea(outer) ;
    areaOfWasher = outerArea -
    innerArea ;
    return  areaOfWasher ;
}
```



## Execution

| main | | |
|---|---|---|
| inner | outer | y |
| 2.0 | 10.0 | |

| WasherArea | | | | |
|---|---|---|---|---|
| inner | outer | innerArea | outerArea | areaOfWasher |
| 2.0 | 5.0 | 12.0 | 75.0 | 63.0 |



## Full Program, Page 2 of 2

```
/* read washer info and print area */
int main(void)
{
    double inner, outer, y ;

    printf ("Input inner radius and outer diameter: ") ;
    scanf (" %lf %lf ", &inner, &outer) ;
    y = WasherArea (inner, outer/2.0) ;

    printf (" %f ", y) ;

    return 0 ;
}
```

## Execution

| main | | |
|---|---|---|
| inner | outer | y |
| 2.0 | 10.0 | 63.0 |

**Output: 63.0**

G2-49

## Functions: Summary

Functions may take several parameters, or none.

Functions may return one value, or none.

Functions are valuable!

- A tool for program structuring.
- Provide *abstract* services: the caller cares what the functions do, but not how.
- Make programs easier to write, debug, and understand.

G2-50

## Looking Ahead

There is still more to learn about functions

- We'll study other methods of parameter passing
- We'll also look at functions as a fundamental design technique

Many students report that functions are the first really difficult concept of the course. They have to be mastered. You haven't seen the last of functions, and you never will!

G2-51