

CSE 142 Computer Programming I

Function Parameters

© 2000 UW CSE

4/11/01

G2-1

Overview

Many concepts in this lecture:

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

Lots of new terminology, too

Near the end of the lecture:

- An extended program, traced

G2-2

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");  
}
```

G2-3

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...

G2-4

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?

G2-5

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

G2-6

```

#include <stdio.h>
int main(void)
{
    PrintBannerLines(5);
    /* produce some output */
    PrintBannerLines(2);
    /* produce final output */
    PrintBannerLines(5);
    return 0;
}

```

Code for PrintBannerLines

The value in the parentheses is called the "argument" to this call.

Code for the Modified Function

The function will start off this way:

```

void PrintBannerLines (int numLines)
{
    ...
}

```

numLines is the "parameter" of the function. numLines 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.)

Passing Results Back to the Caller

Specification: Write a function that, given the radius, computes the area of a circle with that radius.

What should it do with the result?

newArea = CircleArea(1.7);

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

newArea = CircleArea(1.7); return 3.14 * r * r;

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 CircleArea (double r)
{
    return 3.14 * r * r;
}

```

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 (returned value's type): 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

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.\n",
           firstRandom, secondRandom, result);
    return 0;
}
```

G2-14

return in void Functions

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

optional

```
/* print banner line */
void print_banner (void)
{
    printf("*****\n");
    printf("*****\n");
    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 Calculate (void) {...}* could it contain this statement?
return;
4. If a function starts off as *void PrintBankBalance (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

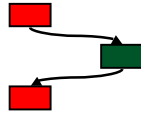
Many people use the term *formal parameter* instead of *parameter* and *actual parameter* instead of *argument*. We will try to stick to *parameter* and *argument* for simplicity, but the other terminology will probably slip in from time to time.

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

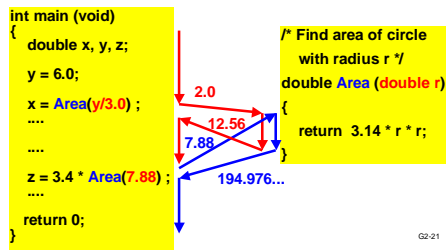
Control and Data Flow

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.

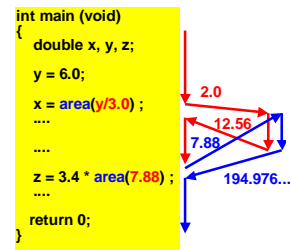
G2-20

Control and Data Flow



G2-21

Control and Data Flow (2)



G2-22

The Way: Function 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.

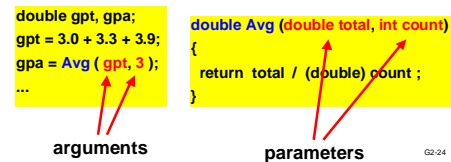
Someone wishing to use a function should be able to hide its body and learn everything they need from the function heading and comment:

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

G2-23

Multiple Parameters

A function may have more than one parameter
Arguments must match parameters in **number**, **order**, and **type**



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

Psst... 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

The Party Line

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 programmed will be used. 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 rsqrd, area;
    rsqrd = r * r;
    area = PI * rsqrd;
    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, area ;

    printf ("Input inner radius and
outer diameter: ");
    scanf ("%f %f", &inner, &outer);
    area = WasherArea (inner, outer/2.0);
    printf ("%f ", area);
    return 0;
}
```

G2-34

Full Program, Page 1 of 2

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

    rsqrd = r * r ;
    area = PI * rsqrd ;
    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, area ;

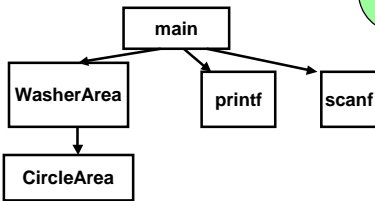
    printf ("Input inner radius and outer diameter: ");
    scanf ("%f %f", &inner, &outer);
    area = WasherArea (inner, outer/2.0);

    printf ("%f ", area);

    return 0;
}
```

G2-36

Showing How Functions are Related



This "static call graph" shows who calls who²⁻³⁷

Local Variables of main

main		
inner	outer	area

G2-38

Parameters and local variables of WasherArea

WasherArea				
inner	outer	innerArea	outerArea	areaOfWasher

G2-39

Parameters and local variables of CircleArea

CircleArea		
r	rsqrd	area

G2-40

Full Program, Page 2 of 2

```

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

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

    printf("%f", area);

    return 0;
}
  
```

G2-41

Full Program, Page 1 of 2

```

#include <stdio.h>
#define PI 3.0

/* 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;
}

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

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

G2-42

Execution

main		
inner	outer	area
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

QOTD: Platonic (Function) Love

Below are descriptions of two functions. Write a (good) function prototype (return type, name, and parameter types/names) for each one.

A function that takes a number and returns the probability that a random integer between 1 and 10 will be greater than that number.

A function that calculates the number of square feet of paint required to completely paint the interior of a room

(No need to write the function body! But... feel free to try if you want extra practice.)

G2-52