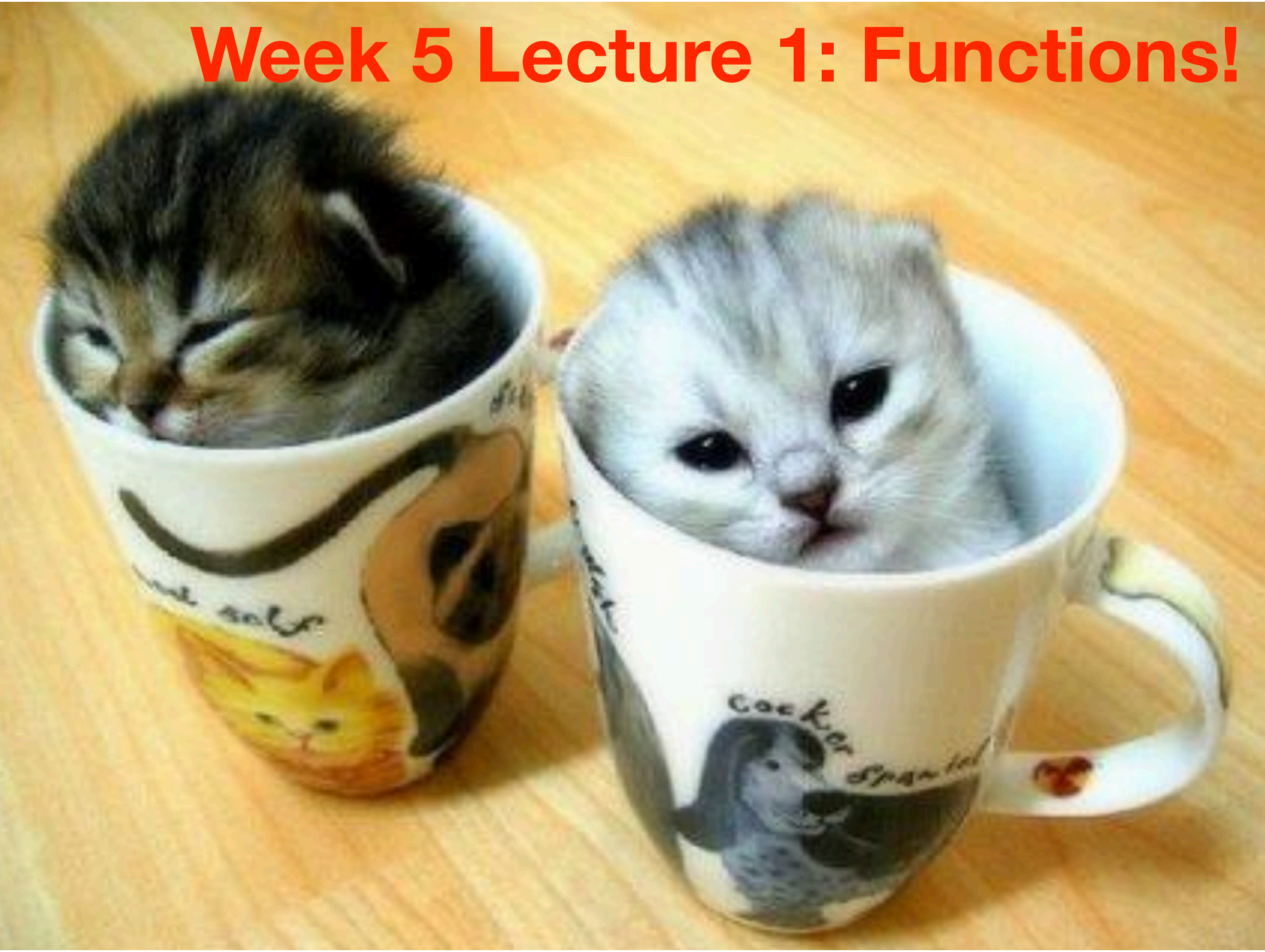
Week 5 Lecture 1: Functions!



Review and Preview

- Best Practice
  - Program formatting
  - Functions
  - More about Functions
  - Assignment 2
  - The Midterm approaches…

K&R: Chapters 1-3
Sections 1.1 to 1.6
Sections 2.1 to 2.6
Sections 3.1 to 3.6

Calling the Function

- To use a function, you have to "**invoke**" it or "**call**" it.
- You do this by using its name and giving it the required parameters and handling the return value (if any).

```
retValue = functionName (param1, param2);
```

- But how does C handle "**passing**" the parameters from one function (*e.g.* `main`) to another one (*e.g.* `functionName`).

Passing Parameters

- There are two ways to pass parameters to a function:
- Pass by **Value**
  - The value of the parameter in the calling routine is copied into the value of the parameter in the function.
  - Changes made to the parameter inside the function have no effect on the value of the variable in the calling routine.

Passing Parameters

- Pass by **Reference**
  - This method tells the function where the actual parameters are in memory.
  - Inside the function, the actual calling routine's variables are used. The variables in the function are the **actual** variables in the calling routine. It does this by means of **pointers**.
  - Any changes made to the parameters in the function affect those variables in the calling routine.

When the function **max** is called
what are the values of **n1** and **n2**?

max function

```
int max ( int *n1, int *n2 ) {
    *n1 = *n1 + 1;
    *n2 = *n2 + 1;
    if ( *n1 > *n2 ) {
        return ( *n1 );
    } else {
        return ( *n2 );
    }
}
……
num1 = 2;
num2 = 4;
maxNum = max(&num1, &num2);
```

Calling the **max** function

What are the values of **num1** and **num2**
after the **max** function has returned?

The Pointer

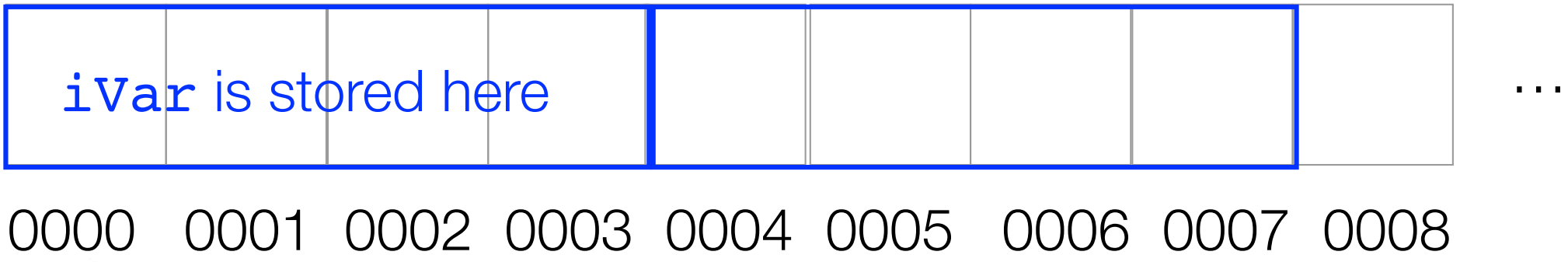
A Short Interlude in our Musings on Functions

What is a Pointer?

- A pointer is a **variable** that contains the **address** of a variable
- **Pointers** and **arrays** are closely related.
- So there is a pointer type and there are operations that manipulate pointers.

What Does a Pointer Point At?

Memory - each byte of memory has an **address**



This is `iPointer`

Want to change the value of `iVar` using `iPointer`?

integer = 4 bytes

```
int *iPointer;
int iVar = 1;
iPointer = &iVar;
```

Pointer Operators

```
int numVar1 = 10;
int numVar2 = 20;
int numArray[3] = { 1, 2, 3 };

int *intPointer;
intPointer = &numVar1;

numVar2 = *intPointer;
intPointer = &numArray[1];
numVar1 = *intPointer;
```

numVar1 = 10
numVar2 = 20
numArray[0] = 1
numArray[1] = 2
numArray[2] = 3

numVar2 = 10
numVar1 = 2

Pointers Declarations

- Every pointer points to a specific data type.

```
int *iPointer;
```

- If `*iPointer` points to the integer `x`, then `*iPointer` can be used anywhere `x` could be used.

```
int x = 10;

*iPointer = x;

*iPointer = *iPointer + 2;
```

What is the value of `x` now?

Pointer Operators

- The unary operator `&` gives the **address** of an object, so the statement

```
pointer = &object;
```

  assigns the address of `object` to the variable `pointer`.

- The `&` operator only applies to variables and array elements.

Pointer Operations

- The unary operator `*` is the **indirection** or **dereferencing** operator.
- When it is applied to a pointer, it accesses the object that the pointer is pointing at.

```
int *iPointer;
int x = 1;
int y = 0;
iPointer = &x;
y = *iPointer;
```

iPointer is pointing to x

y is equal to the value pointed at by iPointer

Binding

```
int iArray[5] = { 1, 3, 5, 7, 9 };

int *iPointer;

int i;

iPointer = &iArray[0];

i = *iPointer;          printf ( “%d\n”, i );

i = *iPointer + 1;      printf ( “%d\n”, i );

i = *(iPointer + 3);    printf ( “%d\n”, i );
```

Pointers and Arrays

- There are two ways to refer to the start of an array:

```
int array[5];

int *pointer;

pointer = &array[0];      OR

pointer = array;
```

```
#include <string.h>
int toUpperCase ( char *line ) {
  int lineLength;
  int character;
  int i;

  lineLength = strlen ( line );
  for ( i=0; i<lineLength; i++ ) {
    character = line[i];
    if (character > 96 && character < 123) {
      line[i] = character - 32;
    }
  }
  return ( 0 );
}
```

Pointers and Passing by Reference (Functions)

- If you want to do either of the following you need to pass by reference:
  - Pass something other than a basic type.
  - Change the contents of the passed variable.
- To pass by reference you send a pointer to the variable to the function.
  - Anything done to the variable in the array will be visible in the calling routine.

Passing by Reference Examples: Arrays

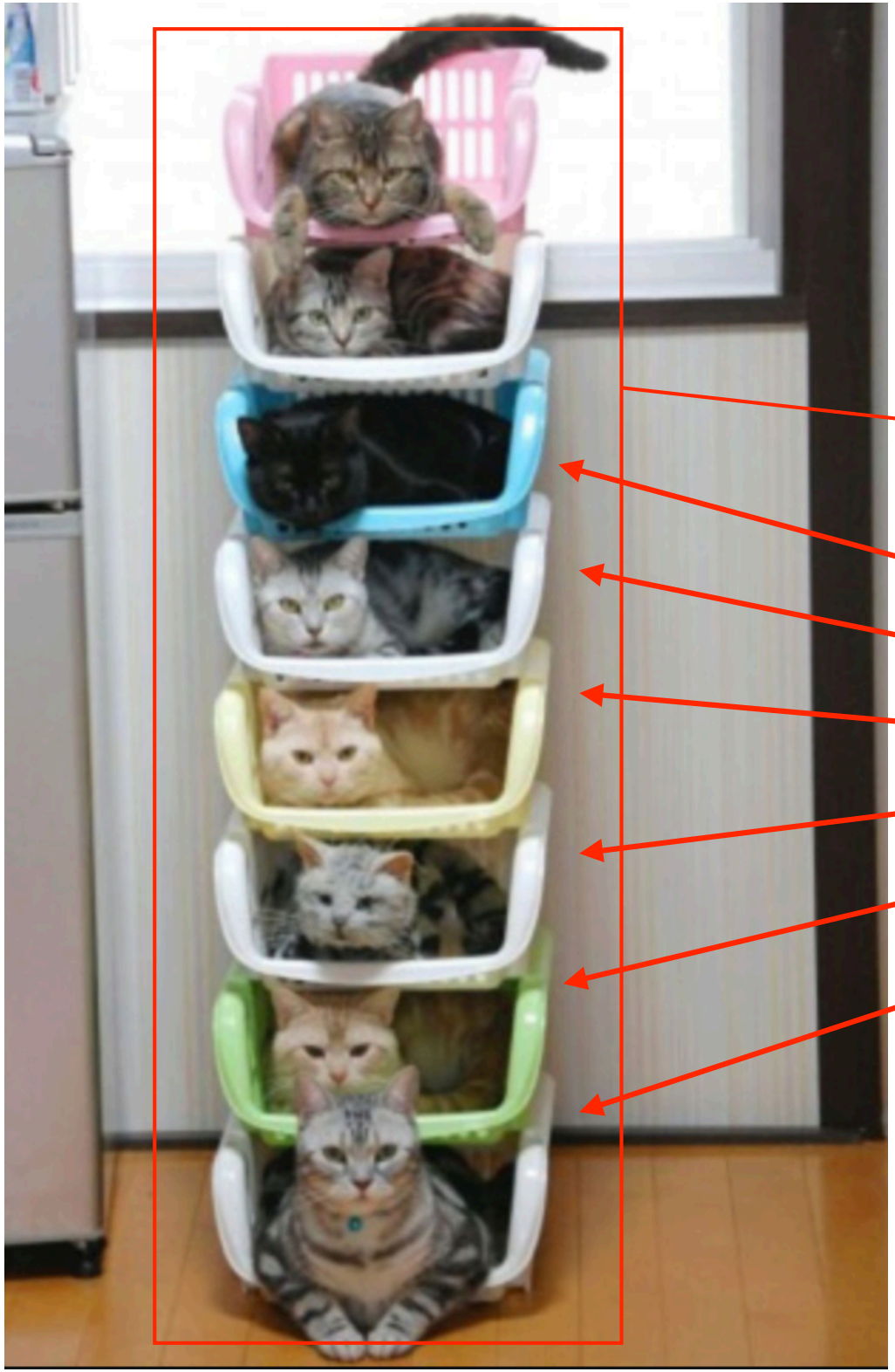
- If you wanted to pass an array of integers to a function than you would first generate a pointer to the array.

```
int array[5];    int *pointer;

pointer = &array[0];
```

- Then you send the pointer to the function

```
funcName ( pointer );

int funcName ( int * );
```

- In the function you treat the pointer as a pointer to an array.



Assignment 2

`findWords.c`

`chop()`
`convertLowerCase()`
`replaceDigits()`
`replacePunc()`
`reduceSpace()`
`trim()`

Specification for `convertUpperCase()` function

- `int convertUpperCase ( char *line );`
- Convert all **alphabetic** characters to **upper** case (do not use the `toupper()` function in the string library for C).
- Returns the number of characters converted from upper case to lower case.

```
#include <string.h>
int convertUpperCase ( char *line ) {
 int lineLength;
 int character;
 int i;
 int total = 0;

 lineLength = strlen ( line );
 for ( i=0; i<lineLength; i++ ){
    character = *(line+i);
    if (character > 96 && character < 123){
       *(line+i) = character - 32;
       total++;
    }
 }
 return ( total );
}
```

line[i] instead of *(line+i)

$ make textRW

```
CC = gcc
CFLAGS = -ansi

textRW: textRW.o
	$(CC) $(CFLAGS) -o textRW textRW.o

textRW.o: textRW.c
	$(CC) $(CFLAGS) -c textRW.c
```

$ cat testFile.txt | ./textRW


NOT THE
MIDTERM !!!!!
CUCUMBERS

Study Hints

- **General Knowledge**
  - Go through the notes, text, and labs and write down anything of interest (definitions, examples, concepts, etc.) on a single sheet of paper - create a cheat sheet that you CANNOT bring to the midterm.
  - Write things out on paper - including programs - this will help you remember.
  - Test yourself - make up questions.

Study Hints

- **Reading Code**
  - Find all the code in the lecture notes, text, labs, and assignment #1 and read in.
    - Read it out loud.
    - Explain the code as you read it.

Study Hints

- **Writing Code**
  - Look at all the code that you have read and think about other code based on this that you could write.
  - Write out test programs by hand, type them in, compile, and test. Repeat.

Midterm Hints

- **Read** the whole exam **first** before you start to answer the questions.
- Answer the questions that are **easy** (you know the answers) **first**.
- Manage your time.
- **READ ALL OF THE QUESTIONS VERY CAREFULLY.**