PURDUE
UNIVERSITY®

CS 240: Programming in C

Lecture 10: Introduction to Pointers

Prof. Jeff Turkstra

Code Style Update

- `#defines` should be at the top beneath #includes
  - We'll start enforcing this for HW4
- Constants should be defined for things that are not immediate and obvious
  - Goal is to improve code readability
  - Eg, columns from HW3

Accommodated Exams

- Please be sure we have a letter from DRC regarding any necessary accommodations
- Accommodated exams will overlap the exam time
  - …in a different room
- We’ll send you an email with more details soon

Final Exam

- Thursday, May 8 10:30am – 12:30pm
  - ELLT 116
- More details later

Next Week

- No Lecture Wednesday 2/26
  - Exam Make-up Day
- I will not be able to hold Thursday and Friday office hours
  - Only Monday
  - Email if you need an appointment (likely virtual)

Midterm Exam 1

- Monday, March 3 8pm – 10pm
  - Look for seating charts in the next week or so – there is more than one room!

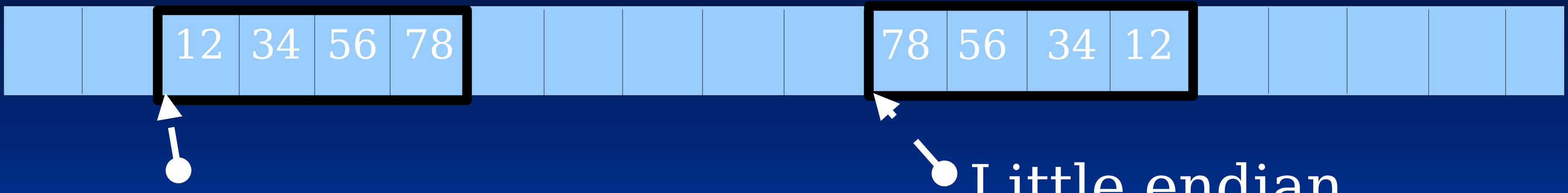
Homework 2

```
596 scores total…
100+:  (0)
 100: ==================== (334)
  90: ===== (71)
  80: === (33)
  70: === (36)
  60: = (14)
  50: = (7)
  40: = (11)
  30: = (4)
  20: = (11)
  10: = (7)
   0: ===== (68)
Average: 79.90
```

Homework 3

```
596 scores total…
100+:  (0)
 100: ===================== (452)
  90: == (35)
  80: == (36)
  70: = (12)
  60: = (8)
  50: = (6)
  40: = (4)
  30: = (2)
  20: = (1)
  10: = (1)
   0: == (39)
Average: 89.90
```

Stored in memory…



- Each box is 1 byte. We can look at it in binary too:

  0x12 = 0b0001 0010
  0x34 = 0b0011 0100
  0x56 = 0b0101 0110
  0x78 = 0b0111 1000

- 0x12345678 =
  0b00010010001101000101011001111000
- 0x78563412 =
  0b01111000010101100011010000010010

Bit setting/clearing

- Use operators to clear/set bits in numbers…

```
int color = 44;  /* binary 00101100 */
int blue = 7;    /* binary 00000111 */

printf(“Color with all blue is %d\n”,
       color | blue);  /* 00101111 */
printf(“Color with no blue is %d\n”,
       color & ~blue);  /* 00101000 */
new_color |= blue & color;
printf(“new_color: %d\n”, new_color);
```

Bit checking

- How can we determine if a specified bit is set (i.e., set to 1)?

```
char bits = 44;  /* binary 00101100 */
char mask = 8;   /* binary 00001000 */

if ((bits & mask) == mask) {
  printf(“The bit is set\n”);
}
else {
  printf(“The bit is cleared\n”);
}
```

A note from our sponsors…

Pointers have been lumped with the **goto** statement as a marvelous way to create impossible-to-understand programs. This is certainly true when they are used carelessly, and it is easy to create pointers that point somewhere unexpected. With discipline, however, pointers can also be used to achieve clarity and simplicity. This is the aspect that we will try to illustrate.

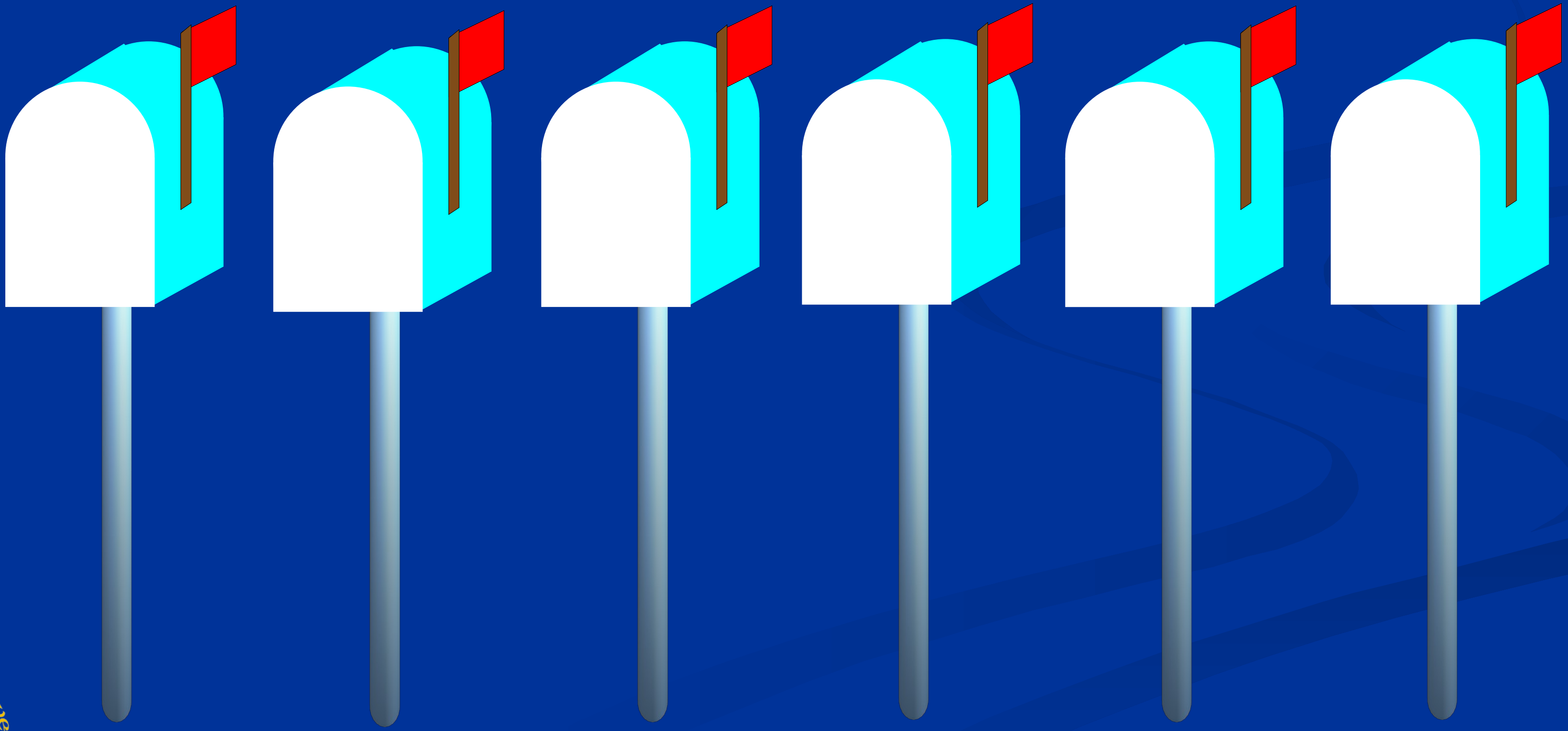
- Brian Kernighan & Dennis Ritchie
  The C Programming Language

Learning about pointers

- Read Chapter 5 of your text. ALL OF IT
  - ...and/or 7 of Beej’s
- Understanding basic pointer concepts is easy
- Understanding the combination of concepts is harder
- Applying the concepts in a meaningful manner takes practice, patience, practice, willingness and especially practice
- We’ll look at one basic concept at a time

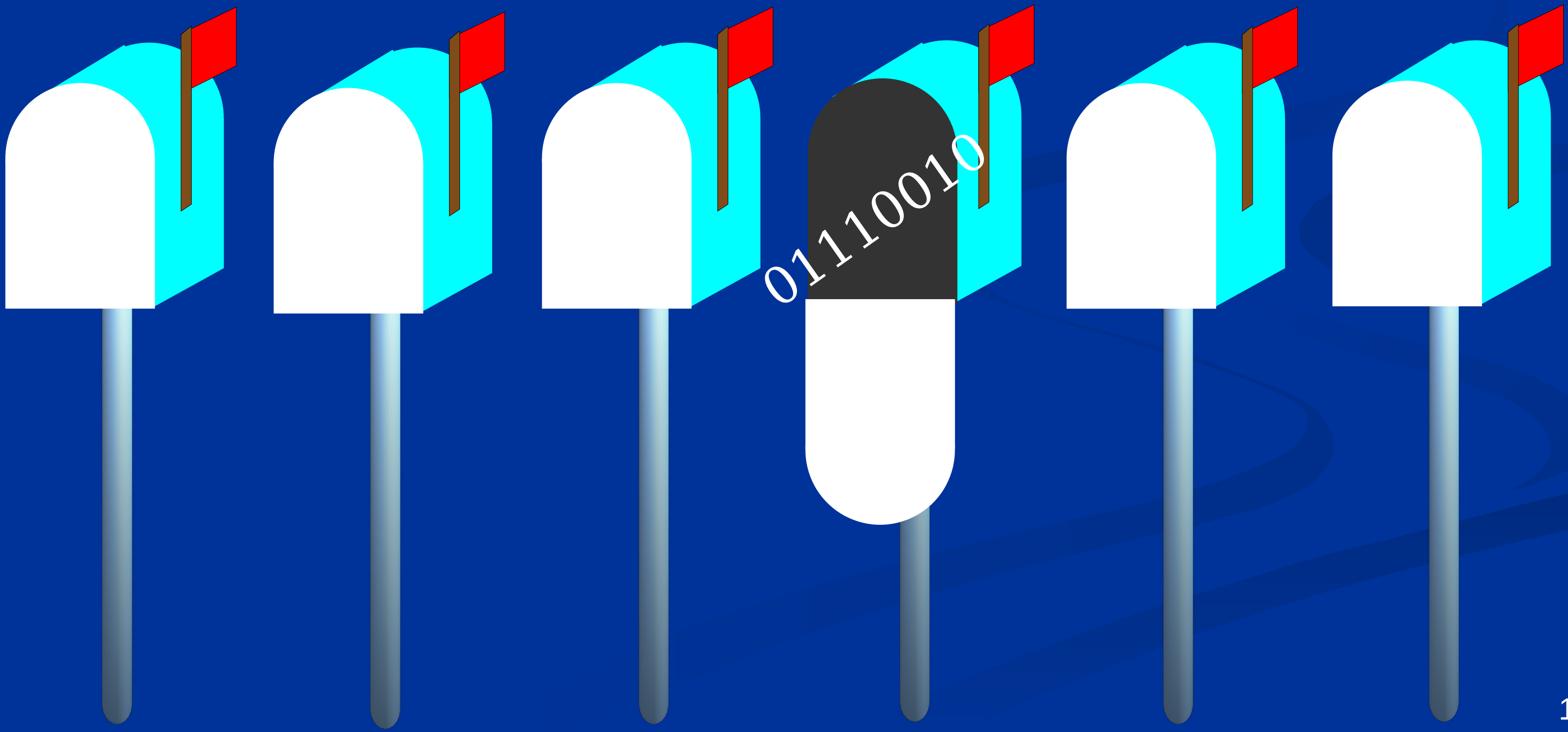
Memory (again)

- Recall that computer memory is organized in a contiguous straight line like a row of mailboxes on a very long road



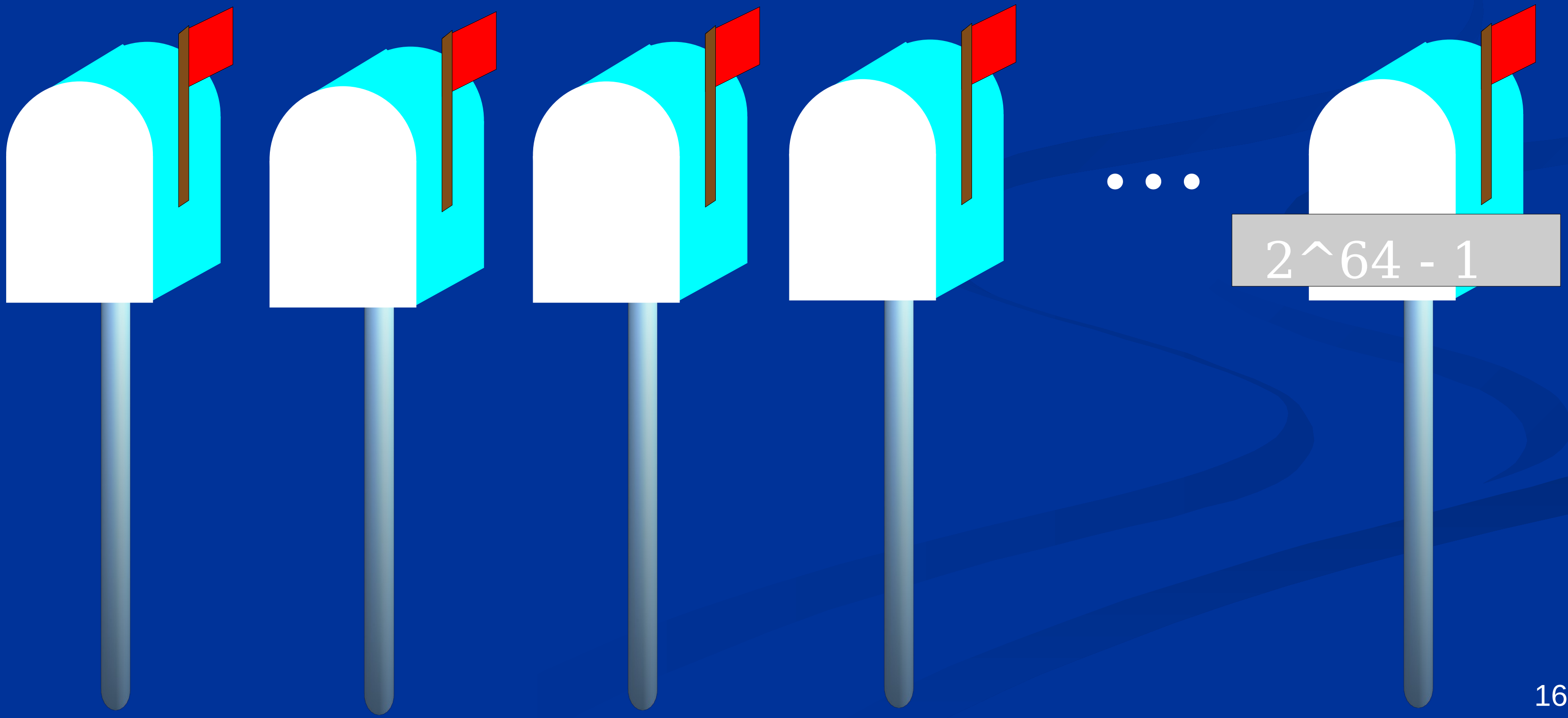
Memory values

- Each mailbox can hold one byte. (8 bits) This can represent a value between 0 and 255 or a single character.
  01110010 = 0x72 = 114 = ‘r’



Memory addresses

- Every mailbox has an address. To put something in a mailbox, you need to know its address. Our addresses go from zero to 18,446,744,073,709,551,615 (2^64 - 1)



What is a pointer?

- Simply, a pointer is a variable that stores the address of another variable

Normal variables

- When we say…
  ```
  int x;
  ```
  the X variable occupies some space in memory.
- When we have a statement like…
  ```
  x = 5;
  ```
  the compiler "knows" how to compute the address of the x variable and can arrange to have a value put there

How can we determine the address of a variable?

- The C language has an operator (&) used to determine the location of any storage element. For example:

  ```
  p = &x;
  ```

  p now holds the address that x resides at
- p is not equivalent to x!
  - Instead, we say that p points to x
- *p is equivalent to x

Address-of…

```
#include <stdio.h>

int main() {
  int x = 0;

  printf(“Address of x: %p\n”, &x);

  return 0;
}
```

How do we define a pointer?

- The creation of a pointer looks like this:

```
int *p;
```

That means that p is a pointer to an integer

How to manipulate a pointer?

- Now that we can find out the address of a variable, how can we use that address?
- The C language has an operator (*) called the indirection operator or a contents-of operator. For example:

```
        *p = 5;
```

  will place the value 5 into the memory location pointed to by p

A complete example

```
#include <stdio.h>

int main() {
  int x = 0;
  int *p = 0;

  p = &x;
  *p = 5;

  printf("x = %d\n", x);
  return 0;
}
```

p now "points" to x

*p is equivalent to x

Why do we need pointers?

- Something we cannot do with variables:

```
#include <stdio.h>
void increment(int n) {
  n = n + 1;
}

int main() {
  int x = 0;
  increment(x);
  printf(“The value of x = %d\n”, x);
  return 0;
}
```

It didn’t work

- Why wasn’t x incremented?
  - x was not passed to the increment function
  - The value of x was passed to the increment function
  - The new value computed inside increment() was not stored back into x
- In fact, there is no way for increment() to modify the value of x
  - We call this “pass-by-value”
- We need some way to tell increment() about the memory location of x. Hmm….

What is a pointer good for?

- Let’s use a pointer, instead…

```
#include <stdio.h>
void increment(int *p) {
  *p = *p + 1;
}

int main() {
  int x = 0;
  increment(&x);
  printf(“The value of x = %d\n”, x);
}
```

Now you understand scanf()

- Now you know why you have to use the ‘&’ operator when you use scanf(). E.g.: `scanf(“%d\n”, &x);`
  - You’re not passing the variable to scanf()
  - You’re telling scanf() what the address of the variable is, so that scanf() can fill it in
- This is called passing by reference, passing by pointer, or passing by address
- Why don’t we need ‘&’ when we pass an array?

Purdue Trivia

- Purdue’s “Big Bass Drum” (or BBD) was commissioned in 1921 by Paul Spotts Emrick
  - Leedy Manufacturing Company
- Ford Model T back axle and wheelbase

Celebrating
Purdue
100
years
REMO

Another pointer example

```
int main() {
  int ctr = 0;
  int *ptr = 0;
  int int4 = 18;
  int int3 = 11;
  int int2 = 10;
  int int1 = 7;

  ptr = &int1;

  for (ctr = 0; ctr < 7; ctr++) {
    printf("Value at address %p: 0x%x (%d)\n",
           ptr, *ptr, *ptr);
    ptr++;
  }

  return 0;
}
```

More pointer basics

- Pointers can be used just as arrays
- Arrays are equivalent to pointers
- “Address-of” (&) can be used on array elements
- …and this is the same as “pointer arithmetic”
- “Address-of” can be used for structs

Pointers can be used as arrays

- When you obtain the address of a variable…

```
        ptr = &x;
```

- You can “dereference” it two ways:

```
    y = *ptr;    /* treat as pointer   */
    z = ptr[0]; /* or 1 element array */
```

- The effect and meaning are exactly the same

Using a pointer as an array

```
#include <stdio.h>

void inc(int *ptr) {
  ptr[0]++; /* use as array */
}

int main() {
  int num = 0;
  inc(&num);  /* pass as a pointer */
  printf(“num = %d\n”, num);
  return 0;
}
```

Arrays are <u>equivalent to</u> pointers

- When you assign an array (not one of its elements) to something, you’re assigning a pointer…

```
    ptr = array;
```

- When you pass an array (not one of its elements) to something, you’re passing a pointer:

```
    strcpy(array1, array2);
```

- When you return an array, you return a pointer…

```
    return array;
```

Example

```
#include <stdio.h>

int *zap(int *ptr) {
  ptr[0] = 0;
  return ptr;
}

int main() {
  int array[100];
  int *ptr = 0;
  ptr = zap(array);
}
```

Differences between arrays and pointers

- You can assign something new to a pointer, but an array always points to the same thing…

```
     ptr = array;    /* OK */
     array = ptr;    /* Not allowed! */
```

- An array definition allocates space for all the elements – but not the "pointer"!
- A pointer definition allocates space only for the pointer value (address).
  - Here, 8 bytes
- A function parameter defined as an array is really just a pointer

For next lecture

- Study the examples in this lecture at home
- Practice the examples
- Modify the examples
- Read Chapter 5 (7 in Beej’s)

Boiler Up!