Arrays & Pointers

CS 3410: Computer System Organization and Programming

Spring 2025

Cornell Bowers CIS
Computer Science

[G. Guidi, A. Sampson, Z. Susag, and H. Weatherspoon]

Administrivia

- **Assignments:**
  - **A0: Infrastructure** due tonight
    - Slip days aren’t tracked
  - **A1: printf** due last night; late due date Sat. (2/1)
    - Slip days *are* tracked
  - **A0/A1 Survey** out now, due Sat.
  - **A2: Minifloat** out today!
    - Due Wed. (2/5)
- **Online Exercises (E0–E4)** due Wed. (2/5)
- **Week 2 TMQ** due Fri. (1/31)

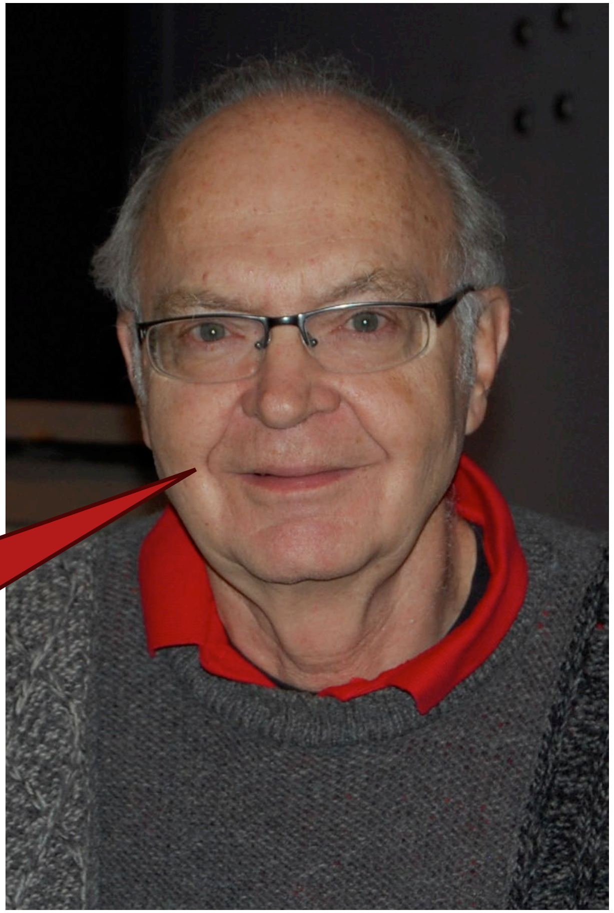
Bit Packing

```c
#include <stdio.h>
#include <stdint.h>
#include <string.h>

int main() {
    uint32_t bits = 0x41040000;
    uint32_t mantissa = bits & 0x007fffff; // mask to isolate mantissa
    uint32_t exponent = (bits & 0x7f800000) >> 23; // bit and bit shift
    uint32_t sign = (bits & 80000000) >> 31; // mask and bit shift

    printf(“s = %b, e = %b, g = %b \n", sign, exponent, mantissa);
    return 0;
}
```

Today’s Plan

- Arrays
- **Pointers**: C’s Central Construct
  - Mental model of memory
  - Pointers as addresses
  - Pointers as references
  - Pointer Arithmetic
  - Arrays as Pointers
  - Fun Pointer Tricks

Donald Knuth



I do consider assignment statements and **pointer variables** to be among computer science’s “most valuable treasures”.

Arrays

Arrays

- An array is a sequence of same-type values that are consecutive in memory
- Fixed-size
  - C does not know the size of an array!

```c
// Declaration
int my_array[4];

// Declaration & Initialization
int my_array[4] = {42, 3, -19, 71};
int my_array[4] = {0};
int my_array[]  = {42, 3, -19, 71};
```

Demo: Arrays

```c
#include <stdio.h>

int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += courses[i];
  }
  printf("the average course is CS %d\n",
         course_total / 7);
  return 0;
}
```

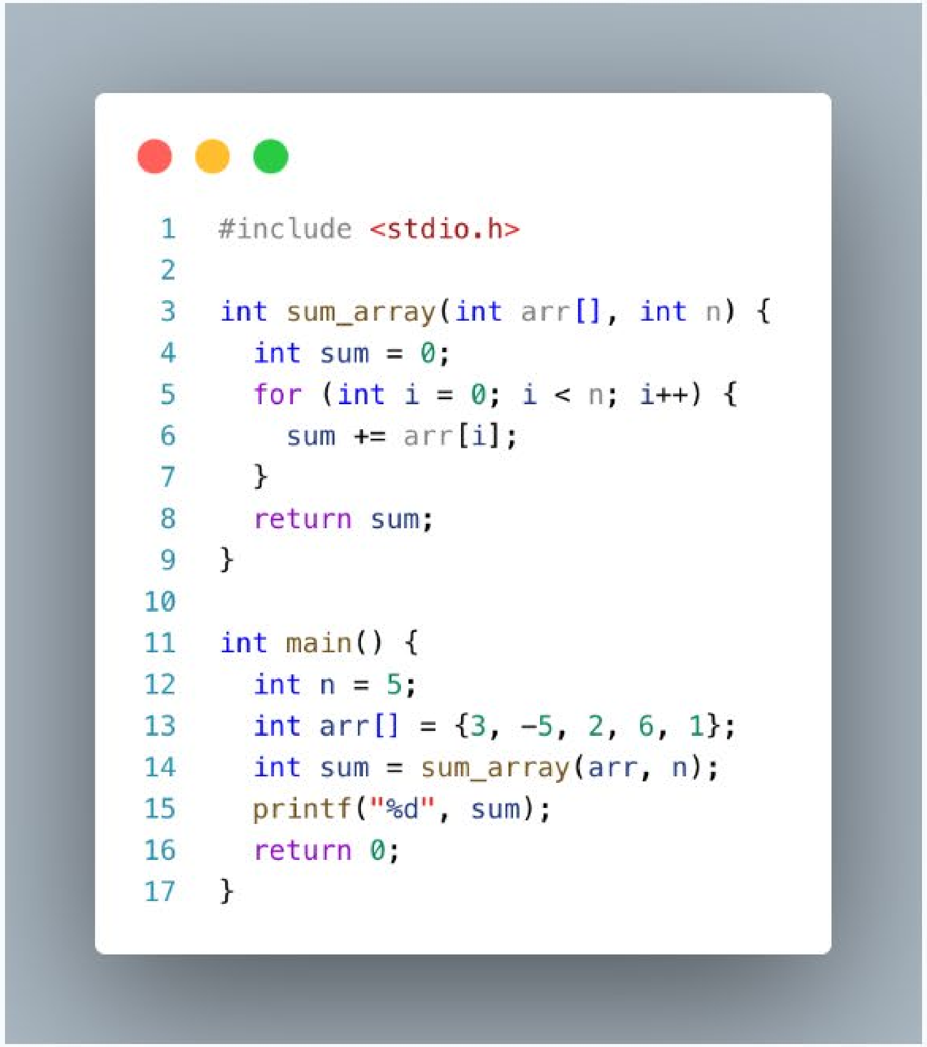
[Arrays and Pointers] sum_array.c

0 surveys completed

0 surveys underway

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

What value does the program print out?



```c
#include <stdio.h>

int sum_array(int arr[], int n) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    sum += arr[i];
  }
  return sum;
}

int main() {
  int n = 5;
  int arr[] = {3, -5, 2, 6, 1};
  int sum = sum_array(arr, n);
  printf("%d", sum);
  return 0;
}
```

7

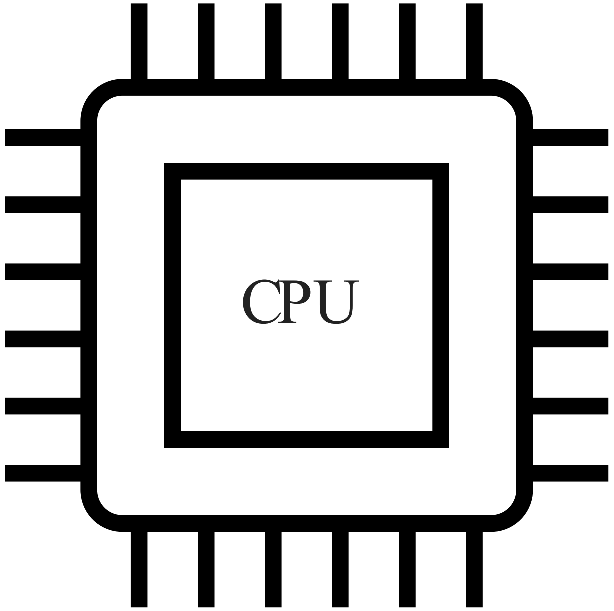
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

Pointers

But first, memory!

Simplified Computer Architecture

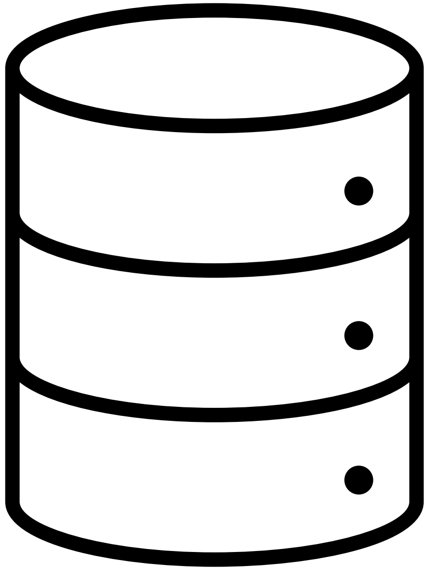
Processor

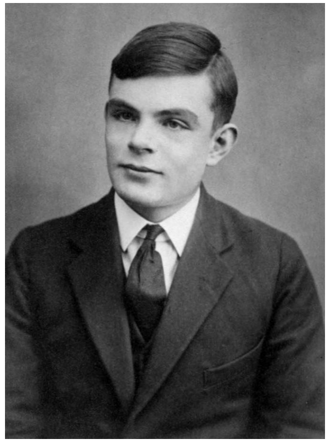


✓ Runs code; does computations

✗ Doesn't remember anything

Memory



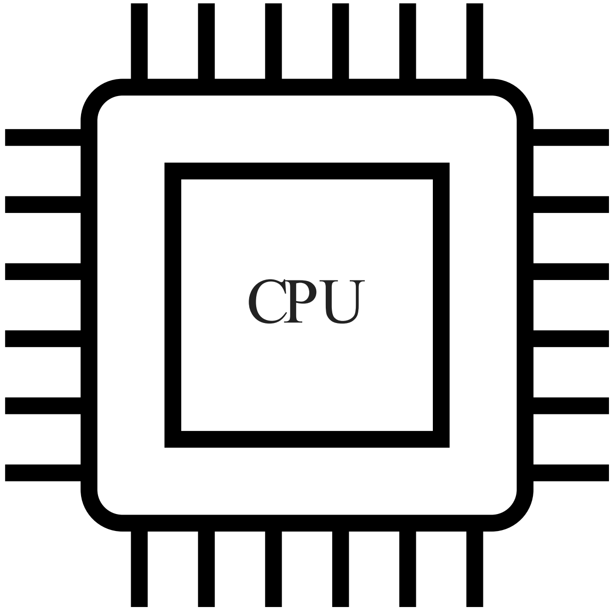


✗ Can't compute anything

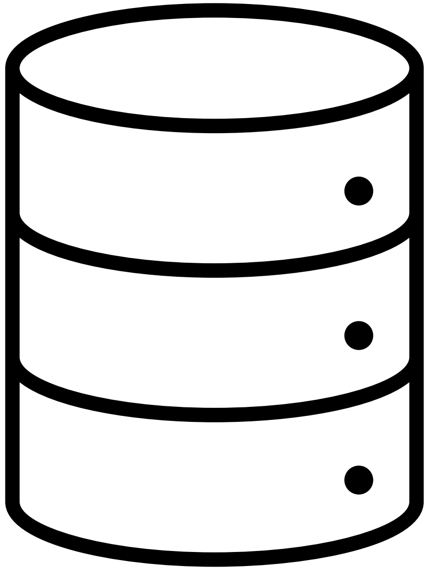
✓ Stores data

A Mental Model of Memory

Processor



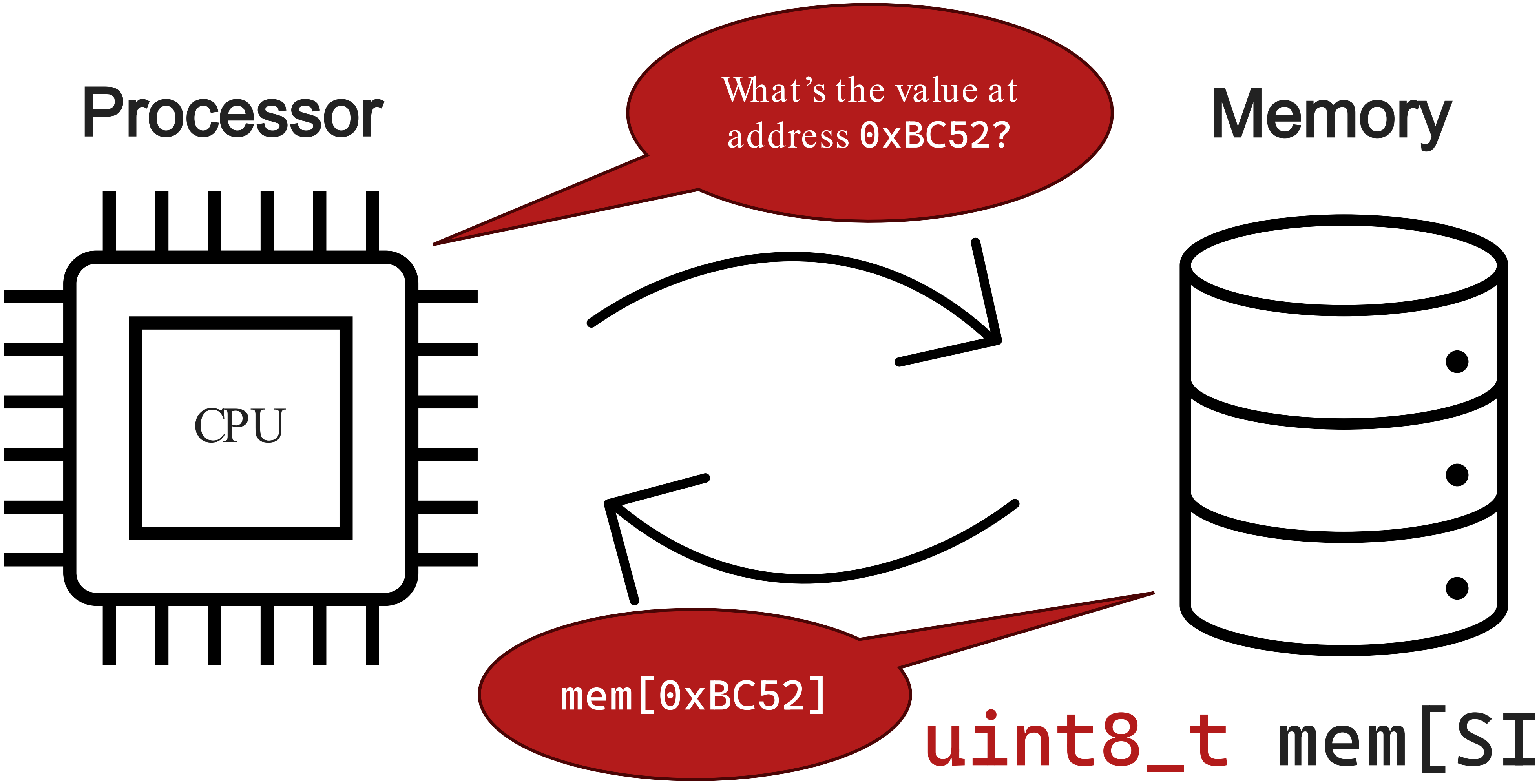
Memory



```
uint8_t mem[SIZE];
```

$$16\text{GB} = 16 \times 1024^3 = 2^4 \times 2^{30} = 2^{34}$$
$$= 17{,}179{,}869{,}184\text{B}$$

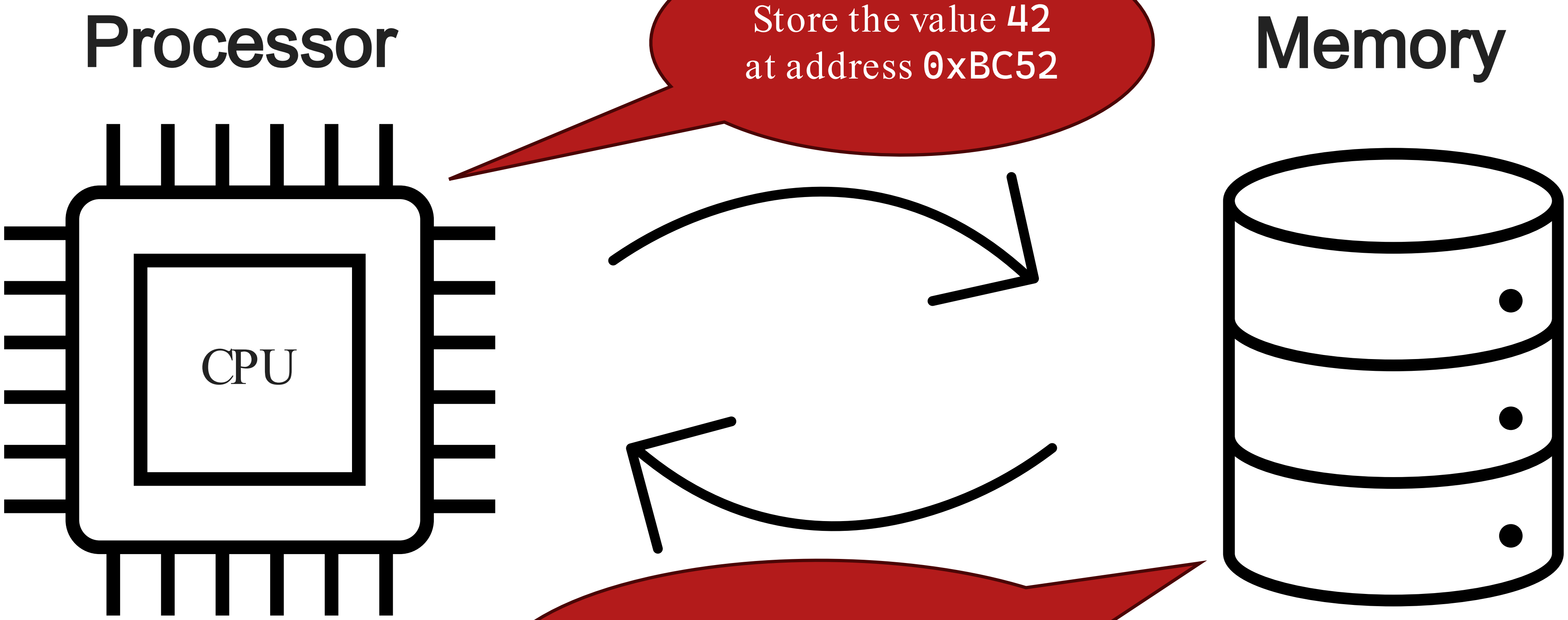
A Mental Model of Memory



```
uint8_t mem[SIZE];
```

$$16\text{GB} = 16 \times 1024^3 = 2^4 \times 2^{30} = 2^{34}$$
$$= 17{,}179{,}869{,}184\text{B}$$

A Mental Model of Memory



Store the value 42
at address 0xBC52

```
mem[0xBC52] = 42;
```

```
8_t mem[SIZE];
```

$16\text{GB} = 16 \times 1024 = 2^4 \times 2^{30} = 2^{34}$
$= 17{,}179{,}869{,}184\text{B}$

Loading a Single Byte

uint8_t mem[SIZE]

Address	Value (uint8_t)
…	…
0xBC52	0xBF
…	…
0x000F	0x02
…	
0x0003	0xEA
0x0002	0x51
0x0001	0xB2
0x0000	0x07

$\texttt{load}_1\texttt{(0xBC52)}$

# of bytes

mem[0xBC52]

Loading Multiple Bytes

$$\mathtt{load}_4(\mathtt{0x0000})$$

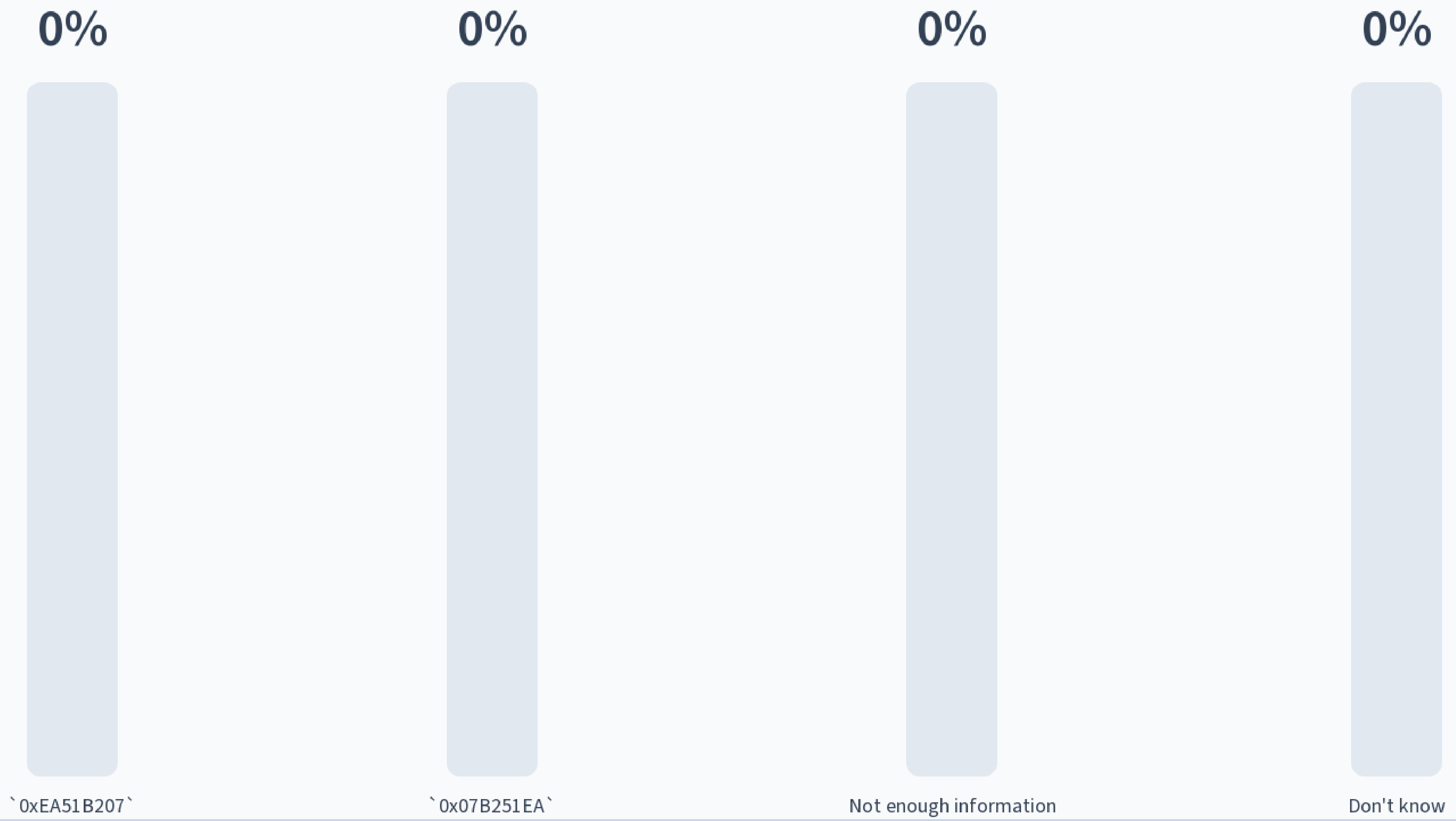
$$=$$

```
uint8_t mem[SIZE]
```

Address	Value (uint8_t)
…	…
0xBC52	0xBF
…	…
0x000F	0x02
…	
0x0003	0xEA
0x0002	0x51
0x0001	0xB2
0x0000	0x07

https://pollev.com/zacharysusag306

What is the 4-byte integer that is loaded from memory address `0x0000`?



Loading Multiple Bytes

Little -Endian

Least *significant byte at the* ***smallest*** *address*

$$\texttt{load}_4(\texttt{0x0000})$$

$$=$$

$$\texttt{0x}\underbrace{\texttt{EA}}\underbrace{\texttt{51}}\underbrace{\texttt{B2}}\underbrace{\texttt{07}}$$

`uint8_t mem[SIZE]`

Address	Value (uint8_t)
…	…
0xBC52	0xBF
…	…
0x000F	0x02
…	
0x0003	0xEA
0x0002	0x51
0x0001	0xB2
0x0000	0x07

Loading Multiple Bytes

Big-Endian

***Most** significant byte at the **smallest** address*

$$\texttt{load}_4(\texttt{0x0000})$$

$$=$$

$$\texttt{0x}\underbrace{\texttt{07}}\underbrace{\texttt{B2}}\underbrace{\texttt{51}}\underbrace{\texttt{EA}}$$

`uint8_t mem[SIZE]`

Address	Value (uint8_t)
…	…
0xBC52	0xBF
…	…
0x000F	0x02
…	
0x0003	0xEA
0x0002	0x51
0x0001	0xB2
0x0000	0x07

A Pointer is An Address

- In C, all data “lives” in memory
  - ⇒ every variable *has an address*
- & AKL@=خJ=>=J=F;=تG>خGH=J9LGJ
  - %=LK9 HGAFL=J بند-A=خ9<<J=KK: LG9 N9JA9: D=

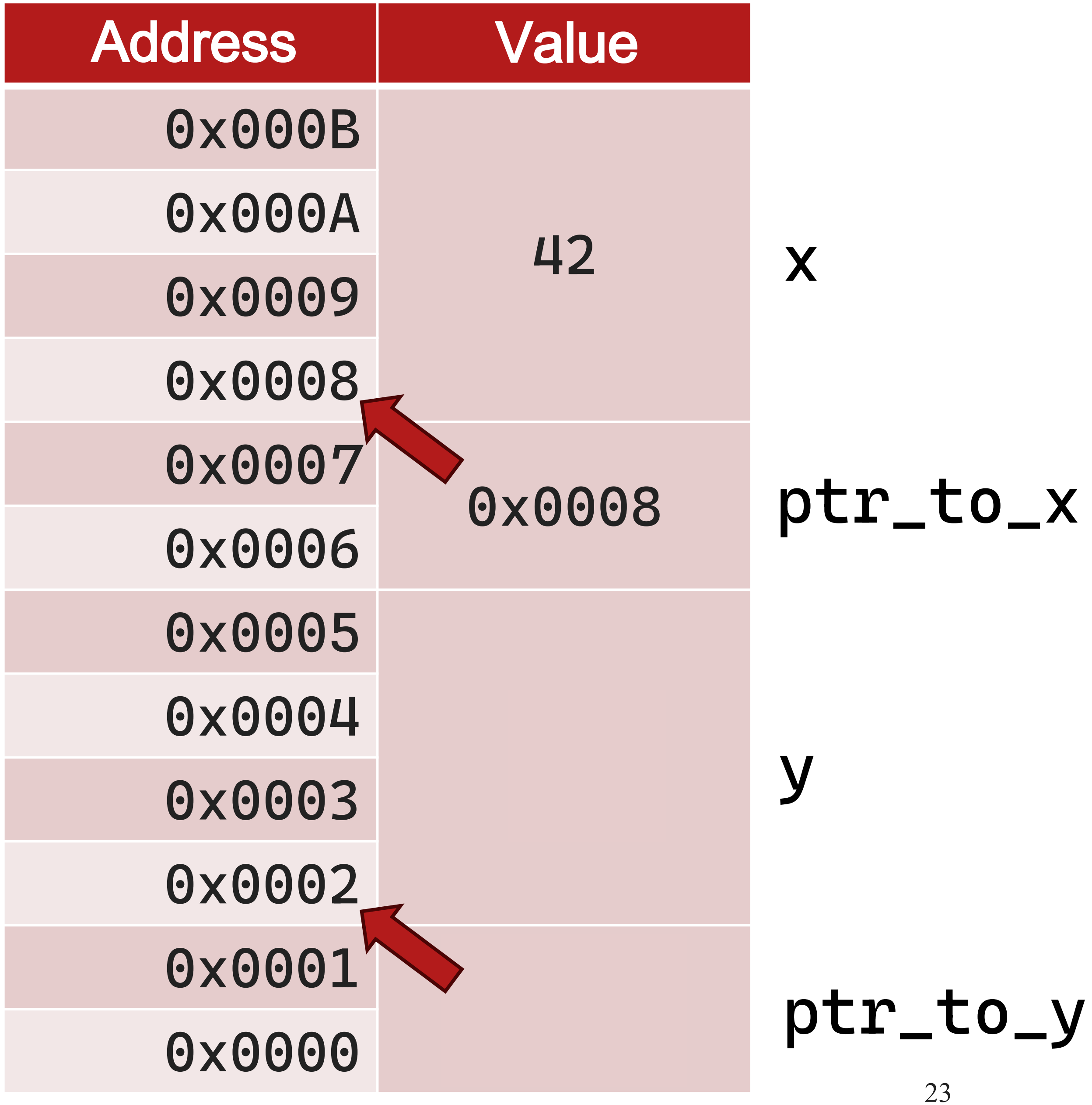
```c
int main() {
  int x = 42;
  int *ptr_to_x = &x;
  printf("x = %d is at %p\n",
         x, ptr_to_x);




  return 0;
}
```

Address	Value	
0x000B		x
0x000A		
0x0009		
0x0008		
0x0007		ptr_to_x
0x0006		
0x0005	?	
0x0004	?	
0x0003	?	
0x0002	?	
0x0001	?	
0x0000	?	

A Pointer is An Address

- In C, all data “lives” in memory
  - ⇒ every variable *has an address*
- & AKL@=ڂJ=>=J=F; =ڎG>ڂGH=J9LGJ
  - %=LK9 HGAFL=J ﯾA=ڂ9<<J=KKﺑ LG9 N9JA9: D=

```
int main() {
  int x = 42;
  int *ptr_to_x = &x;
  printf("x = %d is at %p\n",
         x, ptr_to_x);
  int y = 5;
  int *ptr_to_y = &y;
  printf("y = %d is at %p\n",
         y, ptr_to_y);
  return 0;
}
```



Address	Value	
0x000B	42	x
0x000A		
0x0009		
0x0008		
0x0007	0x0008	ptr_to_x
0x0006		
0x0005		y
0x0004		
0x0003		
0x0002		
0x0001		ptr_to_y
0x0000		

Pointer Types

- Pointers are just addresses to memory
  - Our RISC-V 64 architecture is 64-bit
- The pointer type tells you the type of the value which it points at
  - Pointer to an integer might be `int*`
  - Pointer to a floating-point value might be `float*`
  - Pointer to a character value might be `char*`
- Pointer declaration is whitespace insensitive

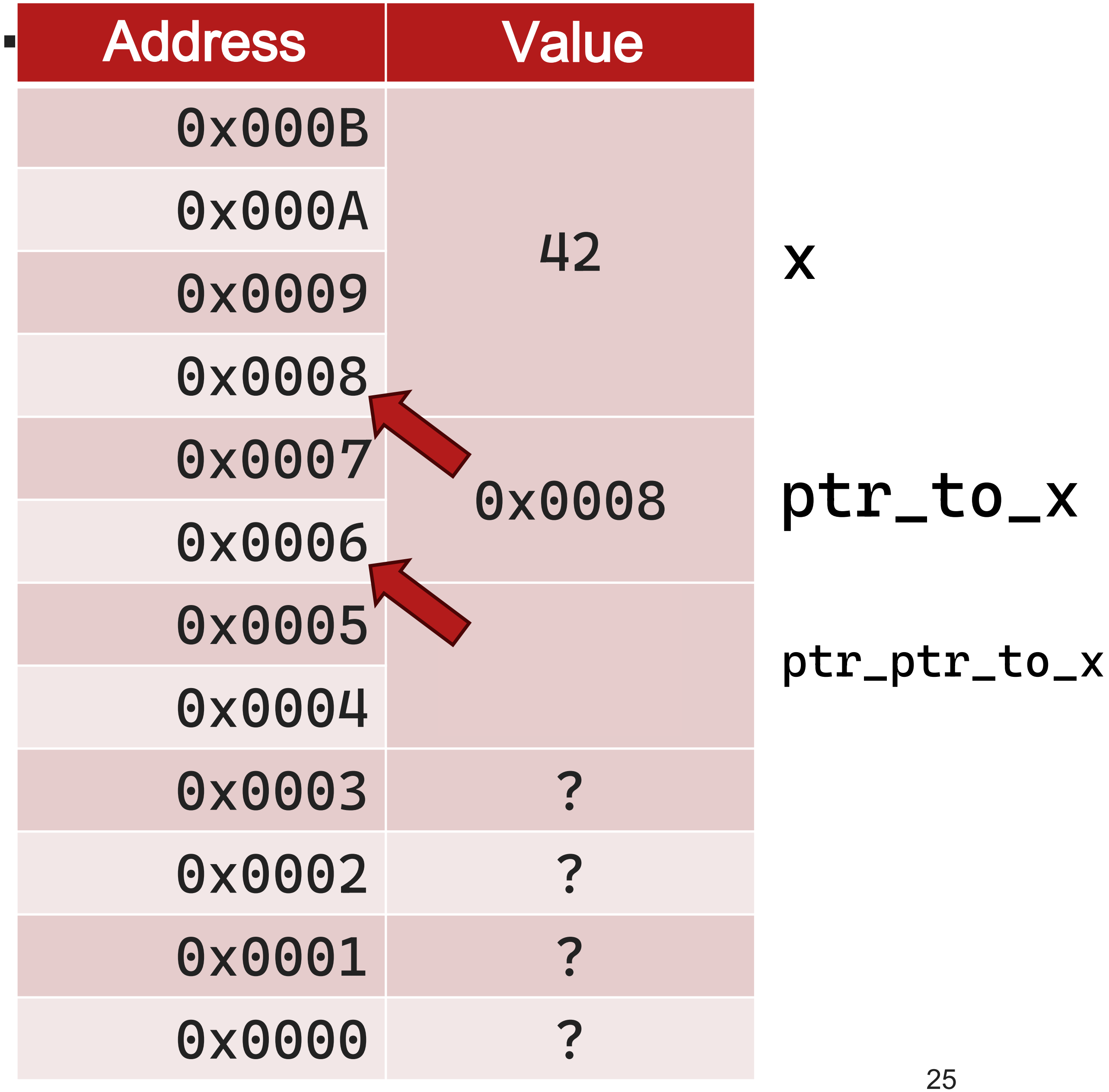
```
int* x;
int *x;
int * x;
```

All still pointers to an `int`!

A pointer to a pointer to…

- #N=F HGAFL=JKDN= AF E =E GJQ…

```
int main() {
  int x = 42;
  int *ptr_to_x = &x;






  return 0;
}
```



Address	Value	
0x000B	42	x
0x000A		
0x0009		
0x0008		
0x0007	0x0008	ptr_to_x
0x0006		
0x0005		ptr_ptr_to_x
0x0004		
0x0003	?	
0x0002	?	
0x0001	?	
0x0000	?	

Cornell Bowers CIS
Computer Science

Pointers are References

- Pointers are *useful* because they are **references**
- * AKL@= <=J=>=J=F; = GH=J9LGJ
  - 3 K=< >GJ DG9<AF? 9F< KLGJAF?

```
int main() {
    int x = 42;
    int *ptr_to_x = &x;
    int x_copy = *ptr_to_x;
    *ptr_to_x = 5;




    return 0;
}
```

Address	Value	
0x000B	5	x
0x000A		
0x0009		
0x0008		
0x0007		ptr_to_x
0x0006		
0x0005		x_copy
0x0004		
0x0003		
0x0002		
0x0001	?	
0x0000	?	

Demo: Pointers as References

```c
#include <stdio.h>

int main() {
  int x = 34;
  int y = 10;

  int *ptr = &x;

  printf("0: x = %d and y = %d and ptr = %p\n", x, y, ptr);
  *ptr = 41;
  printf("1: x = %d and y = %d and ptr = %p\n", x, y, ptr);
  ptr = &y;
  printf("2: x = %d and y = %d and ptr = %p\n", x, y, ptr);
  *ptr = 20;
  printf("3: x = %d and y = %d and ptr = %p\n", x, y, ptr);

  return 0;
}
```

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

https://pollev.com/zacharysusag306

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```c
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value
0x000B	
0x000A	
0x0009	
0x0008	
0x0007	
0x0006	
0x0005	
0x0004	
0x0003	
0x0002	
0x0001	
0x0000	

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```c
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	0	a
0x000A		
0x0009		
0x0008		
0x0007		
0x0006		
0x0005		
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```c
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	0	a
0x000A	1	b
0x0009		
0x0008		
0x0007		
0x0006		
0x0005		
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
    uint8_t a = 0;
    uint8_t b = 1;
    uint8_t *p = &a;
    uint8_t *q = &b;
    uint8_t **r = &p;
    **r = 10;
    *r = q;
    *p = 11;
    return 0;
}
```

Address	Value	
0x000B	0	a
0x000A	1	b
0x0009	0x000B	p
0x0008		
0x0007		
0x0006		
0x0005		
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	0	a
0x000A	1	b
0x0009	0x000B	p
0x0008		
0x0007	0x000A	q
0x0006		
0x0005		
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```c
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	0	a
0x000A	1	b
0x0009	0x000B	p
0x0008		
0x0007	0x000A	q
0x0006		
0x0005	0x0008	r
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
➡ **r = 10;
  *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	10	a
0x000A	1	b
0x0009	0x000B	p
0x0008		
0x0007	0x000A	q
0x0006		
0x0005	0x0008	r
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
➡ *r = q;
  *p = 11;
  return 0;
}
```

Address	Value	
0x000B	10	a
0x000A	1	b
0x0009	0x000A	p
0x0008		
0x0007	0x000A	q
0x0006		
0x0005	0x0008	r
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Poll Everywhere

What are the values of:

1. a
2. b
3. *p
4. *q
5. **r

```
int main() {
  uint8_t a = 0;
  uint8_t b = 1;
  uint8_t *p = &a;
  uint8_t *q = &b;
  uint8_t **r = &p;
  **r = 10;
  *r = q;
➡ *p = 11;
  return 0;
}
```

Address	Value	
0x000B	10	a
0x000A	11	b
0x0009	0x000A	p
0x0008		
0x0007	0x000A	q
0x0006		
0x0005	0x0008	r
0x0004		
0x0003		
0x0002		
0x0001		
0x0000		

Arrays as Pointers

An array is asequenceof same-type values that are consecutive in memory.

```
int main() {
  int arr[3] = {42, -839, 1000};

  printf("first  element is at %p\n",
         &arr[0]);
  printf("second element is at %p\n",
         &arr[1]);
  printf("third  element is at %p\n",
         &arr[2]);
  return 0;
}
```

Address	Value	
0x000B		arr[2]
0x000A		
0x0009		
0x0008		
0x0007		arr[1]
0x0006		
0x0005		
0x0004		
0x0003		arr[0]
0x0002		
0x0001		
0x0000		

Arrays as Pointers

An array is asequenceof same-type values that are consecutive in memory.

```c
int main() {
  int arr[3] = {42, -839, 1000};

  printf("first  element is at %p\n",
         &arr[0]);
  printf("second element is at %p\n",
         &arr[1]);
  printf("third  element is at %p\n",
         &arr[2]);
  return 0;
}
```

Address	Value	
0x000B	1000	arr[2]
0x000A		
0x0009		
0x0008		
0x0007	-839	arr[1]
0x0006		
0x0005		
0x0004		
0x0003	42	arr[0]
0x0002		
0x0001		
0x0000		

Formula for address of an element at index i

Base Address (i.e., address of first element)

Index

$$b + s \cdot i$$

Size of elements, in bytes

Arrays as Pointers to the First Element

```c
#include <stdio.h>

int main() {
  int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};

  printf("first element is at %p\n", &courses[0]);
  printf("the array itself is %p\n", courses);

  return 0;
}
```

`courses` and `&courses[0]` point to the same address!

Passing Arrays to Functions

```c
int sum_n(int *vals, int count) {
  int total = 0;
  for (int i = 0; i < count; ++i) {
    total += vals[i];
  }
  return total;
}
int main() {
  int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
  int sum = sum_n(courses, 7);
  printf("the average course is CS %d\n",
             sum / 7);
  return 0;
}
```

- C does not store the length of an array!
  - You must pass the length alongside the array

Pointer Arithmetic

Question:
Can we compute addresses ourselves?

```c
void experiment(int* courses) {
  printf("courses     = %p\n", courses);
  printf("courses + 1 = %p\n", courses + 1);
}

int main() {
  int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
  experiment(courses);
  return 0;
}
```

```
$ ./a.out
courses     = 0x1555d56bb0
courses + 1 = 0x1555d56bb4
```

Pointer Arithmetic Rule

- In C, pointer arithmetic “moves” pointers by *elementsized chunks*
  - Element size is determined by pointer type
- `courses` has type `int*`
  - Element size is 4 bytes
- Example:
  - `courses + n` 9<<K 4 × n : QL=KLG9<<J=KKG> `courses`

Dereferencing Elements of an Array

```c
void experiment(int* courses) {
  printf("courses[0] = %d\n", *(courses + 0));
  printf("courses[5] = %d\n", *(courses + 5));
}

int main() {
  int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
  experiment(courses);
  return 0;
}
```

```
$ ./a.out
courses[0] = 1110
courses[5] = 3110
```