Resources

RISC-V Infrastructure

RISC-V Infrastructure

Tools

- Unix Shell
- Git
- SSH
- Makefiles

C Programming

- Compilation
- Language Basics
- Basic Types
- Prototypes & Headers
- Control Flow
- Declared Types
- Bit Packing
- Pointers
- Arrays
- Strings
- Macros
- Memory Allocation

RISC-V Assembly

- RISC-V Assembly
- RISC-V ISA Reference

Computer System Organization and Programming

The resources page on our website has background on many lecture topics.

Review: Floating-Point Numbers

CS 3410: Computer System Organization and Programming

[K. Bala, A. Bracy, E. Sirer, Z. Susag, H. Weatherspoon, D. Garcia, and K. Laeufer]

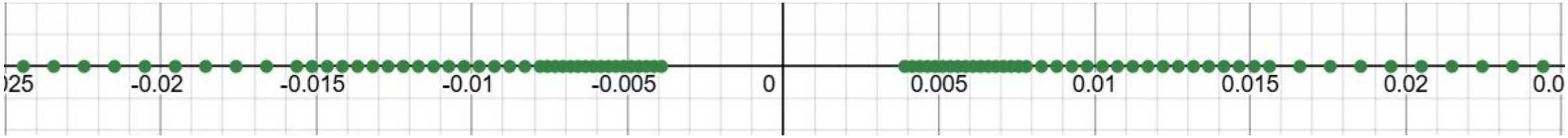
Decoding IEEE 754 style floating-point numbers

- e = all one → g = 0 → $(-1)^{\mathbf{s}}$ x Infinity
  → g ≠ 0 → NaN (Not a Number)
- e > 0 → $(-1)^{\mathbf{s}}$ x 1.**g** x $2^{\mathbf{e} - B}$
- e = 0 → $(-1)^{\mathbf{s}}$ x 0.**g** x $2^{-(B-1)}$ [float.exposed]
- B = $2^{\#\text{ of exponent bits} - 1} - 1$

?	? ? ? ?	? ? ? ? ? ?
Sign **s**	Exponent **e**	Significand **g**

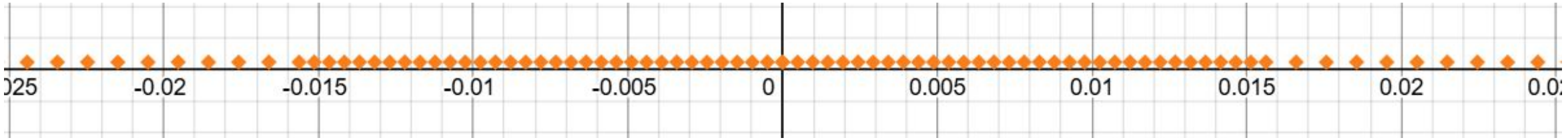
Subnormals and the smallest (non-zero) number

Without subnormals: $(-1)^{\mathbf{s}} \times 1.\mathbf{0} \times 2^{\mathbf{0}\ -\ B}$



With subnormals: $(-1)^{\mathbf{s}} \times 1.\mathbf{0} \times 2^{\mathbf{1}\ -\ B}$

$(-1)^{\mathbf{s}} \times 0.\mathbf{0000000000000001} \times 2^{\mathbf{1}\ -\ B}$



CS3410 minifloat

- $(-1)^{\mathbf{s}} \times \mathbf{g} \times 2^{-3} \times 2^{\mathbf{e}\,-\,3}$

? ? ? ? ? ? ? ? ? ? ? ?

Sign **s** Exponent **e** Significand **g**

Pointers & Arrays

CS 3410: Computer System Organization and Programming

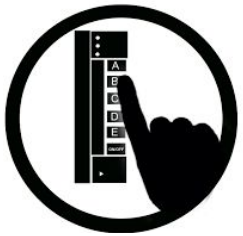
Spring 2025

Pointers as References

```
a = {'x': 0}
b = a
b['x'] += 10
print(f"a = {a}, b = {b}")
```

```
python ref1.py
```

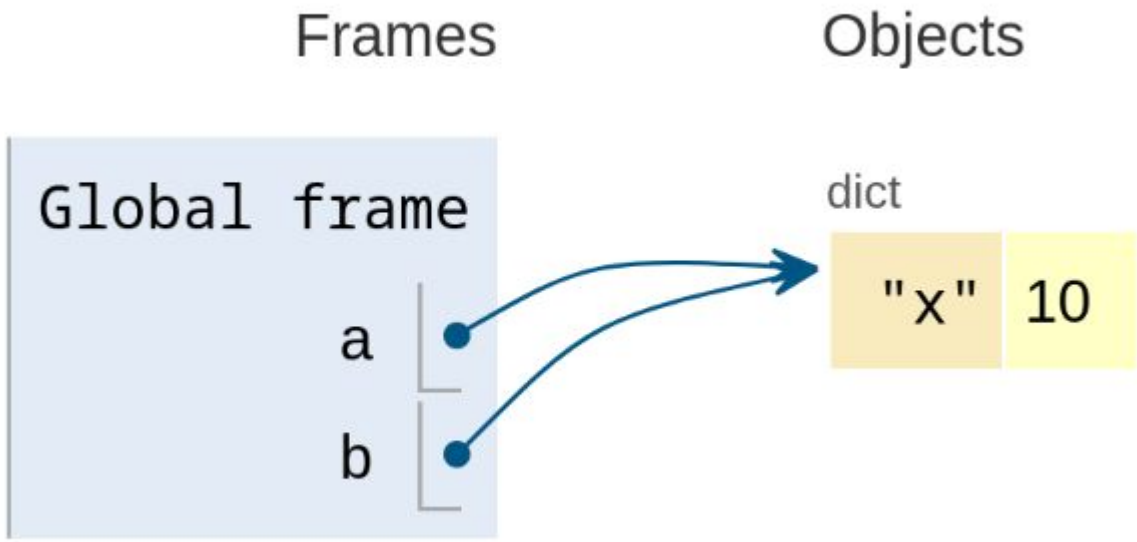
PollEv.com/cs3410



Pointers as References

```
a = {'x': 0}
b = a
b['x'] += 10
print(f"a = {a}, b = {b}")
```

```
python ref1.py
```



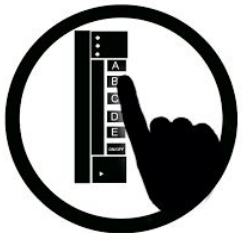
Executed with: https://pythontutor.com

Pointers as References

```
a = 0
b = a
b += 10
print(f"a = {a}, b = {b}")
```

```
python ref2.py
```

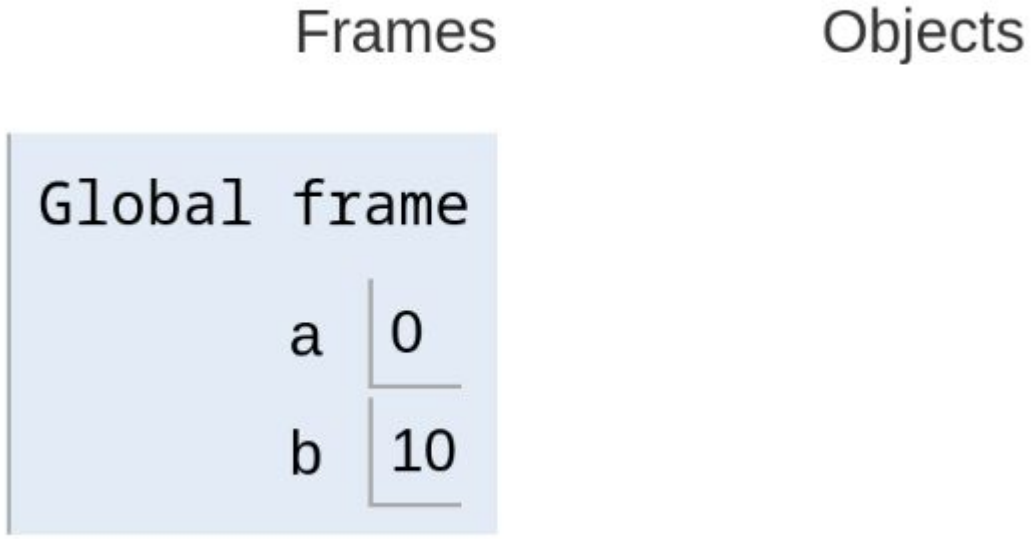
PollEv.com/cs3410



Pointers as References

```
a = 0
b = a
b += 10
print(f"a = {a}, b = {b}")
```

```
python ref2.py
```

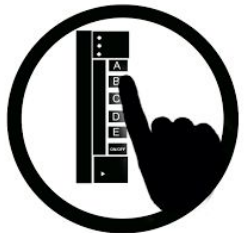


Executed with: https://pythontutor.com

Pointers as References

```
public static void main(String[] args){
    int a = 0;
    int b = a;
    b += 10;
    System.out.println("a = " + a + ", b = " + b);
}
```

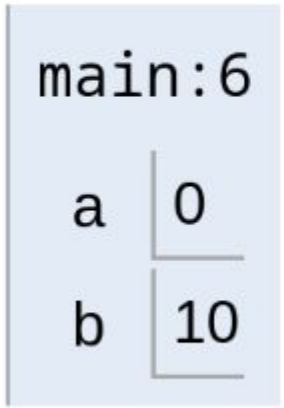
```
javac ref1.java && java ref
```



PollEv.com/cs3410

Pointers as References

```
public static void main(String[]
    int a = 0;
    int b = a;
    b += 10;
    System.out.println("a = " + a
}
```



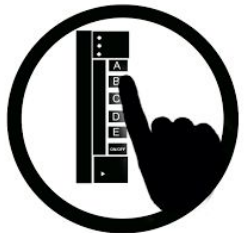

```
javac ref1.java && java ref
```

Executed with: https://pythontutor.com

Pointers as References

```
public static void main(String[] args){
    int[] d = new int[1];
    int[] e = d;
    e[0] += 10;
    System.out.println(
        "d[0] = " + d[0] + ", e[0] = " + e[0]);
}
```

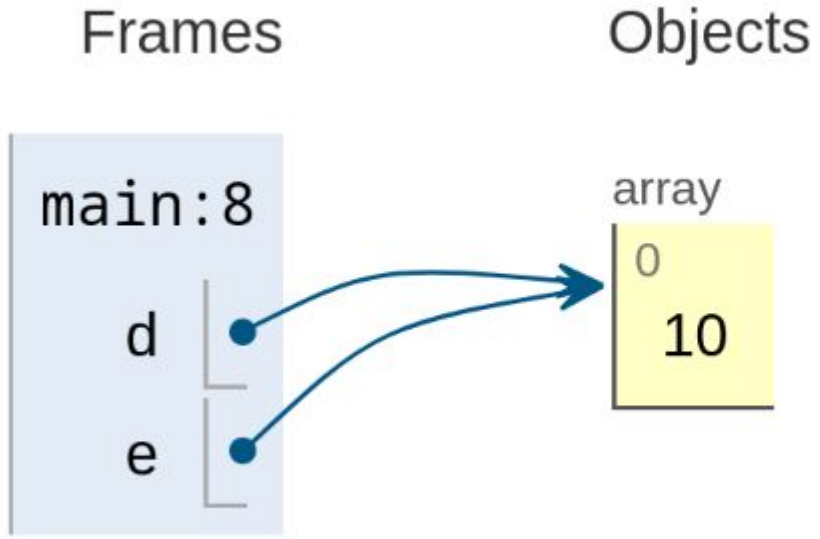
```
javac ref2.java && java ref
```



PollEv.com/cs3410

Pointers as References

```
public static void main(String[] args){
    int[] d = new int[1];
    int[] e = d;
    e[0] += 10;
    System.out.println(
        "d[0] = " + d[0] + ", e
}
```



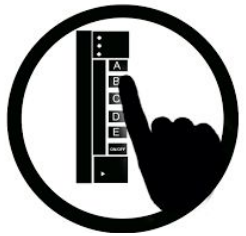

```
javac ref2.java && java ref
```

Executed with: https://pythontutor.com

Pointers as References

```c
int main() {
    int a  = 0;
    int b = a;
    b += 10;
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

```
gcc ref1.c && ./a.out
```

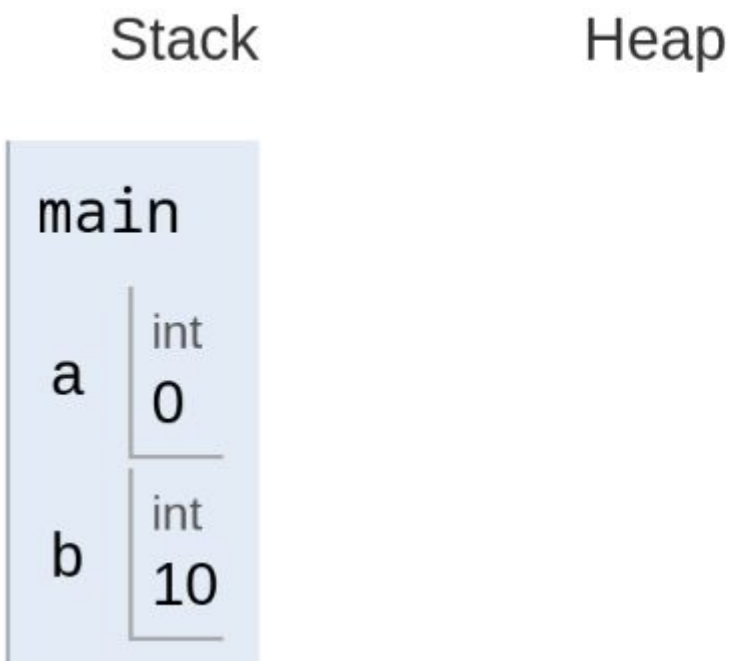


PollEv.com/cs3410

Pointers as References

```c
int main() {
    int a  = 0;
    int b = a;
    b += 10;
    printf("a = %d, b = %d\n",
    return 0;
}
```

```
gcc ref1.c && ./a.out
```

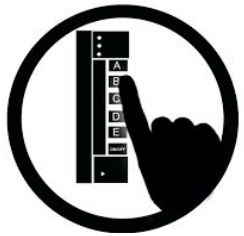


Executed with: https://pythontutor.com

Pointers as References

```c
int main() {
    int *d = malloc(sizeof(int));
    *d = 0; // C arrays are uninitialized by default
    int *e = d;
    *e += 10;
    printf("*d = %d, *e = %d\n", *d, *e);
    return 0;
}
```

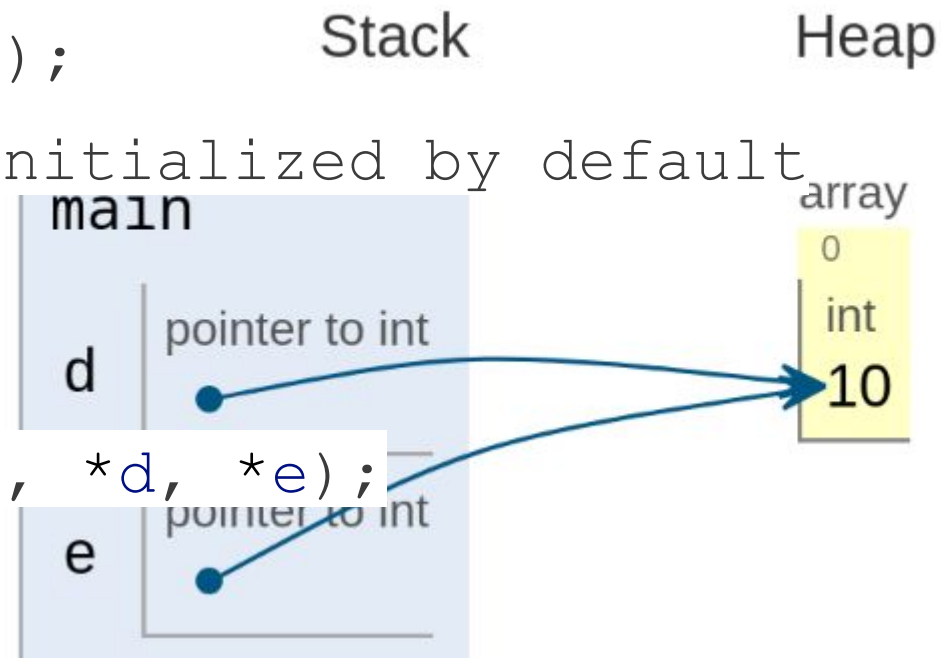
```
gcc ref2.c && ./a.out
```



PollEv.com/cs3410

Pointers as References

```
int main() {
    int *d = malloc(sizeof(int));
    *d = 0; // C arrays are uninitialized by default
    int *e = d;
    *e += 10;
    printf("*d = %d, *e = %d\n", *d, *e);
    return 0;
}
```




```
gcc ref2.c && ./a.out
```

Executed with: https://pythontutor.com

Pointers as References

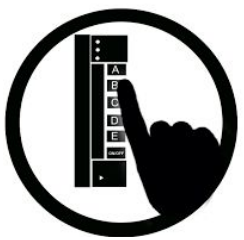
```
a = 0
inc1(a)
print(f"a = {a}")

a = [0]
inc2(a)
print(f"a = {a}")
```

```
def inc1(a):
    a += 1

def inc2(a):
    a[0] += 1
```

PollEv.com/cs3410



python ref3.py

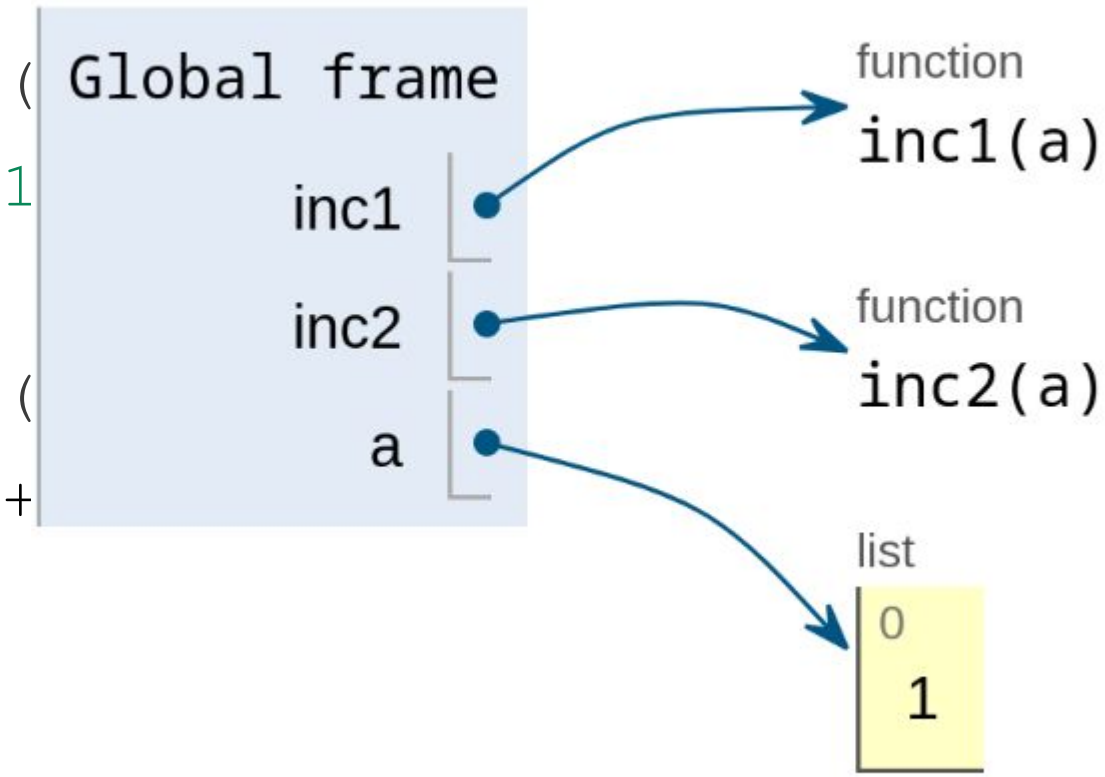
Pointers as References

```
a = 0
inc1(a)
print(f"a = {a}")


a = [0]
inc2(a)
print(f"a = {a}")
```

```
def inc1(
    a += 1

def inc2(
    a[0] +
```




```
python ref3.py
```

Executed with: https://pythontutor.com
Run this one on your own, interactively, to understand the solution.

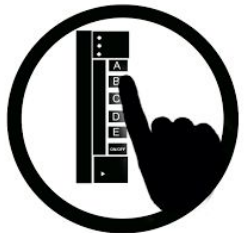
Pointers as References

```
public static void main(String[] args){
    int a = 0;
    inc1(a);
    System.out.println("a = " + a);
    int[] d = new int[1];
    inc2(d);
    System.out.println("d[0] = " + d[0]);
}
```

```
static void inc1(int a ) {
    a += 1;
}

static void inc2(int[] a) {
    a[0] += 1;
}
```

javac ref3.java && java ref



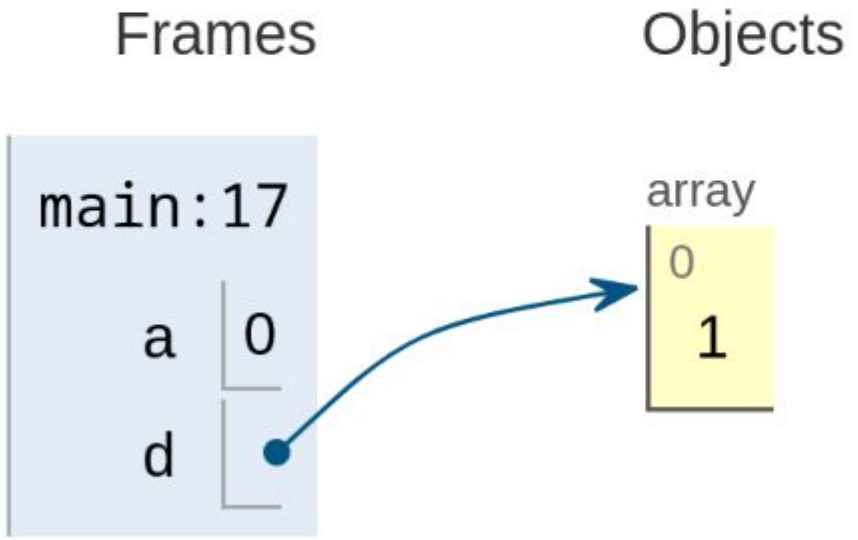
PollEv.com/cs3410

Pointers as References

```
public static void main(String[] args)
    int a = 0;
    inc1(a);
    System.out.println("a = " + a);
    int[] d = new int[1];
    inc2(d);
    System.out.println("d[0] = " + d[0]
}
```

Print output (drag lower right corner to resize)

```
a = 0
d[0] = 1
```




```
javac ref3.java && java ref
```

Executed with: https://pythontutor.com
Run this one on your own, interactively, to understand the solution.

Pointers as References

```c
int main() {
    int a = 0;
    inc1(a);
    printf("a = %d\n", a);
    inc2(&a);
    printf("a = %d\n", a);
    inc3(&a);
    printf("a = %d\n", a);
    return 0;
}
```

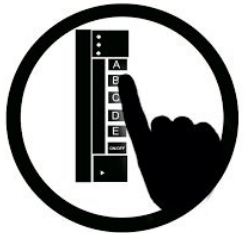
```c
void inc1(int a) {
    a += 1;
}

void inc2(int *a) {
    a += 1;
}

void inc3(int *a) {
    *a += 1;
}
```

```
gcc ref3.c && ./a.out
```

PollEv.com/cs3410

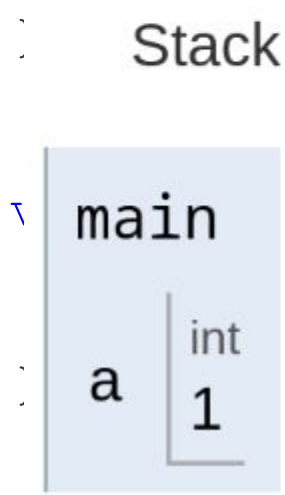


Pointers as References

```c
int main() {
    int a = 0;
    inc1(a);
    printf("a = %d\n", a);
    inc2(&a);
    printf("a = %d\n", a);
    inc3(&a);
    printf("a = %d\n", a);
    return 0;
}
```

Print output (drag lower right corner to resize)

```
a = 0
a = 0
a = 1
```




```
gcc ref3.c && ./a.out
```

Executed with: https://pythontutor.com
Run this one on your own, interactively, to understand the solution.

Pointers as References

- Java and Python use references for arrays, lists, dictionaries, objects, etc.
- Java and Python treat “primitives” like `int` as values.

Pointers as References

- Java and Python use references for arrays, lists, dictionaries, objects, etc.
- Java and Python treat “primitives” like `int` as values.
- In C, the programmer can treat objects of every type either as a value or as a reference.

Pointers as References

- Java and Python use references for arrays, lists, dictionaries, objects, etc.
- Java and Python treat "primitives" like `int` as values.
- In C, the programmer can treat objects of every type either as a value or as a reference.
- Must use special syntax to distinguish:
  - **&** creates a reference
  - ***** refers to value pointed to by reference

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;
}
```

1110
1111
2110
2112
2800
3110
3410

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;
}
```

1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```c
#include <stdio.h>
int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  int *course_ptr = courses;
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += *(course_ptr + i);

  }
  printf("the average course is CS %d\n", course_total / 7);
  return 0;
}
```

1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```c
#include <stdio.h>
int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  int *course_ptr = &courses[0];
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += *(course_ptr + i);

  }
  printf("the average course is CS %d\n", course_total / 7);
  return 0;
}
```

1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```c
#include <stdio.h>
int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  int *course_ptr = &courses[0];
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += *course_ptr;
    course_ptr += 1;
  }
  printf("the average course is CS %d\n", course_total / 7);
  return 0;
}
```

1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```c
#include <stdio.h>
int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  char *course_ptr = (char *)courses;
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += *((int *)course_ptr);
    course_ptr += 1;
  }
  printf("the average course is CS %d\n", course_total / 7);
  return 0;
}
```

1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```
#include <stdio.h>
```

array.c:8:21: runtime error: load of misaligned address 0x7b3cf8b09021 for type 'int', which requires 4 byte alignment

```
                                                    7);
}
```

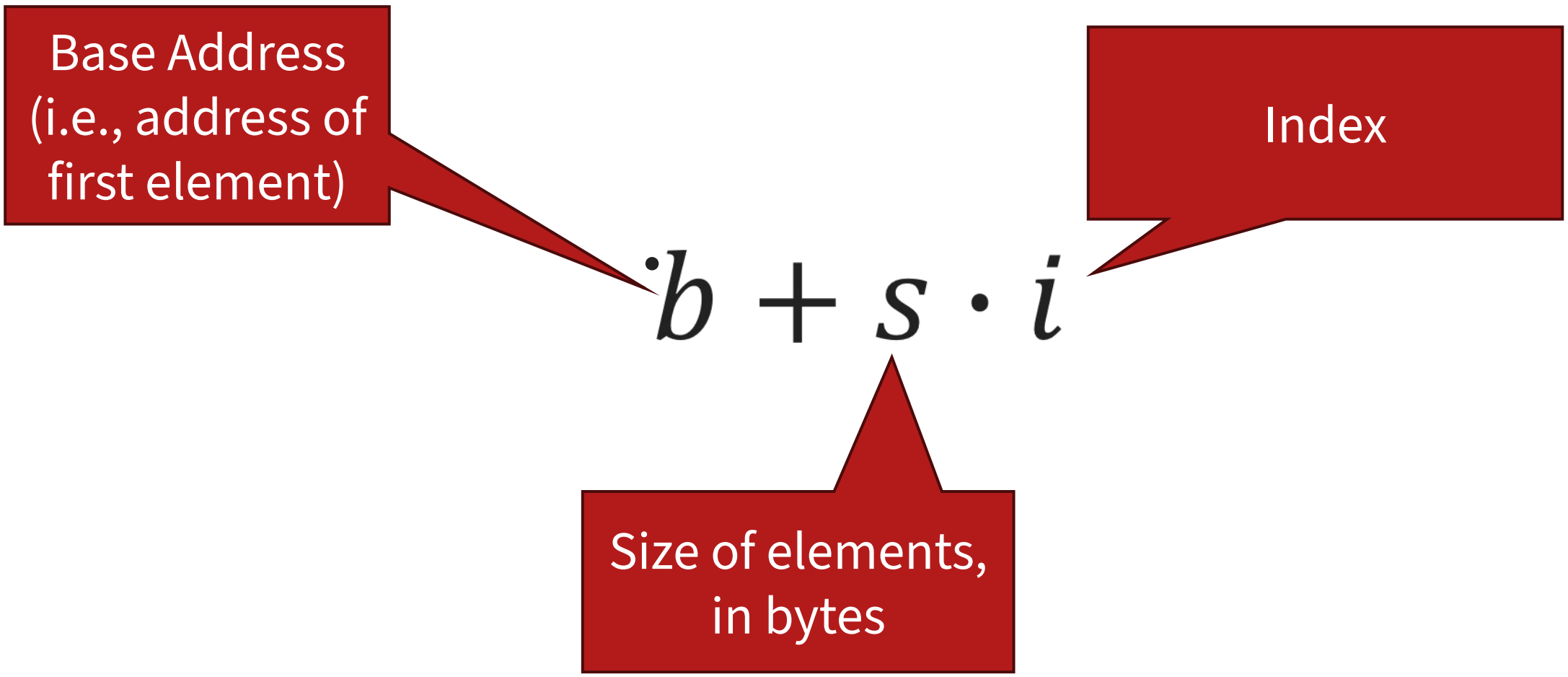
1110
1111
2110
2112
2800
3110
3410

Demo: Arrays

```c
#include <stdio.h>
int main() {
  int courses[7] = {1110, 1111, 2110,
                    2112, 2800, 3110, 3410};
  char *course_ptr = (char *)courses;
  int course_total = 0;
  for (int i = 0; i < 7; ++i) {
    course_total += *((int *)course_ptr);
    course_ptr += sizeof(int);
  }
  printf("the average course is CS %d\n", course_total / 7);
  return 0;
}
```

1110
1111
2110
2112
2800
3110
3410

Formula for address of an element at index i



Passing Arrays to Functions

```
1  int sum_n(int *vals, int count) {
2    int total = 0;
3    for (int i = 0; i < count; ++i) {
4      total += vals[i];
5    }
6    return total;
7  }
8  int main() {
9    int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
10   int sum = sum_n(courses, 7);
11   printf("the average course is CS %d\n",
12          sum / 7);
13   return 0;
14 }
```

- 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?

```
1  void experiment(int* courses) {
2    printf("courses     = %p\n", courses);
3    printf("courses + 1 = %p\n", courses + 1);
4  }
5
6  int main() {
7    int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
8    experiment(courses);
9    return 0;
10 }
```

Pointer Arithmetic

Question:
Can we compute addresses ourselves?

```
1  void experiment(int* courses) {
2    printf("courses     = %p\n", courses);
3    printf("courses + 1 = %p\n", courses + 1);
4  }
5
6  int main() {
7    int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
8    experiment(courses);
9    return 0;
10 }
```

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

Pointer Arithmetic Rule

- In C, pointer arithmetic "moves" pointers by *element-sized chunks*
  - Element size is determined by pointer type
- `courses` has type `int*`
  - Element size is 4 bytes
- **Example:**
  - `courses + n` adds $4 \times n$ bytes to address of `courses`

Dereferencing Elements of an Array

```
1  void experiment(int* courses) {
2    printf("courses[0] = %d\n", *(courses + 0));
3    printf("courses[5] = %d\n", *(courses + 5));
4  }
5
6  int main() {
7    int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
8    experiment(courses);
9    return 0;
10 }
```

Dereferencing Elements of an Array

```
1  void experiment(int* courses) {
2    printf("courses[0] = %d\n", *(courses + 0));
3    printf("courses[5] = %d\n", *(courses + 5));
4  }
5
6  int main() {
7    int courses[7] = {1110, 1111, 2110, 2112, 2800, 3110, 3410};
8    experiment(courses);
9    return 0;
10 }
```

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

Cornell Bowers CIS
Computer Science

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a = 0;
    int b = 1;
    printf("%d\n", &a > &b);
    return 0;
}
```

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a = 0;
    int b = 1;
    printf("%d\n", &a > &b);
    return 0;
}
```

No!

Comparing pointers to objects that are not part of the same struct or array is undefined.

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[1]);
    return 0;
}
```

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[1]);
    return 0;
}
```

Yes!

Pointers to elements of the same array.

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[4]);
    return 0;
}
```

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[4]);
    return 0;
}
```

Yes!

Pointers to elements of the same array and one past the array.

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[5]);
    return 0;
}
```

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    printf("%d\n", &a[0] > &a[5]);
    return 0;
}
```

No!

Not allowed to compare a pointer that points more than one past the array.

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    void* b = (void*)a;
    b += 4;
    printf("%d\n", *((int*)b));
    return 0;
}
```

Pointer Arithmetic: Is this allowed?

```c
#include<stdio.h>
int main() {
    int a[4] = {0, 1, 2, 3};
    void* b = (void*)a;
    b += 4;
    printf("%d\n", *((int*)b));
    return 0;
}
```

No!

Not allowed to perform arithmetic on void* pointer.

Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

`main` frame

Address	Value
&a	
&b	
&p	
&q	
&r	

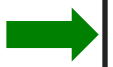
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

`main` frame

Address	Value
&a	0
&b	
&p	
&q	
&r	

Pointer Diagrams



```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

`main` frame

Address	Value
&a	0
&b	1
&p	
&q	
&r	

Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

`main` frame

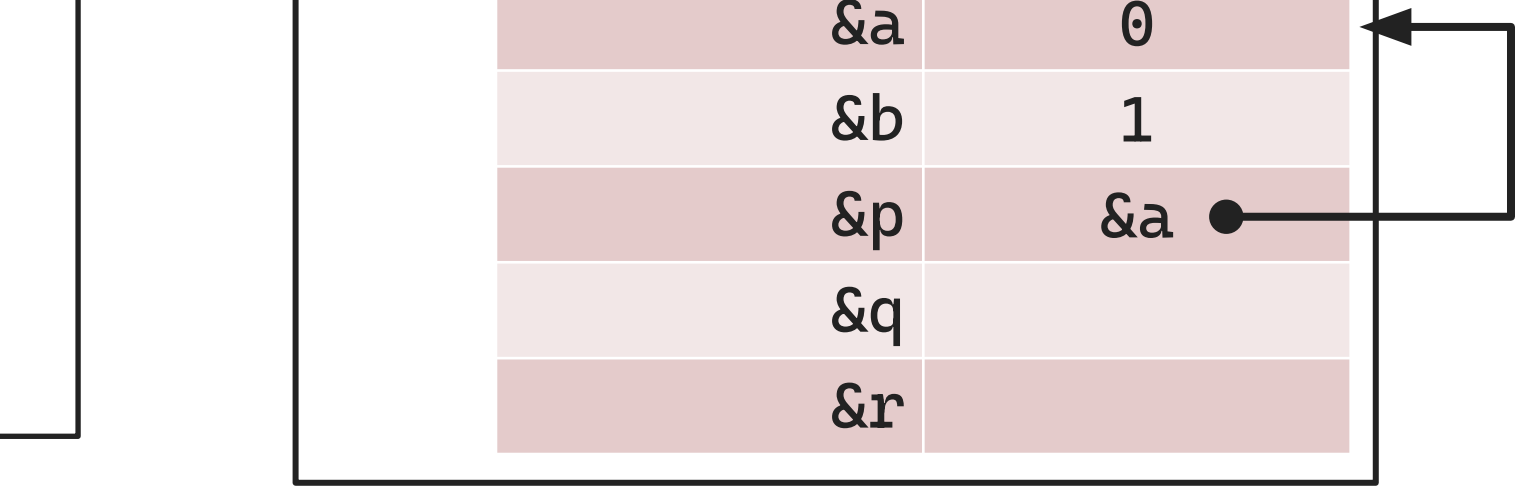
Address	Value
&a	0
&b	1
&p	&a
&q	
&r	

Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

`main` frame

Address	Value
&a	0
&b	1
&p	&a
&q	
&r	

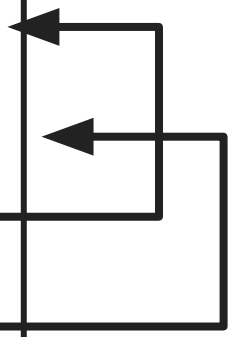


Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```

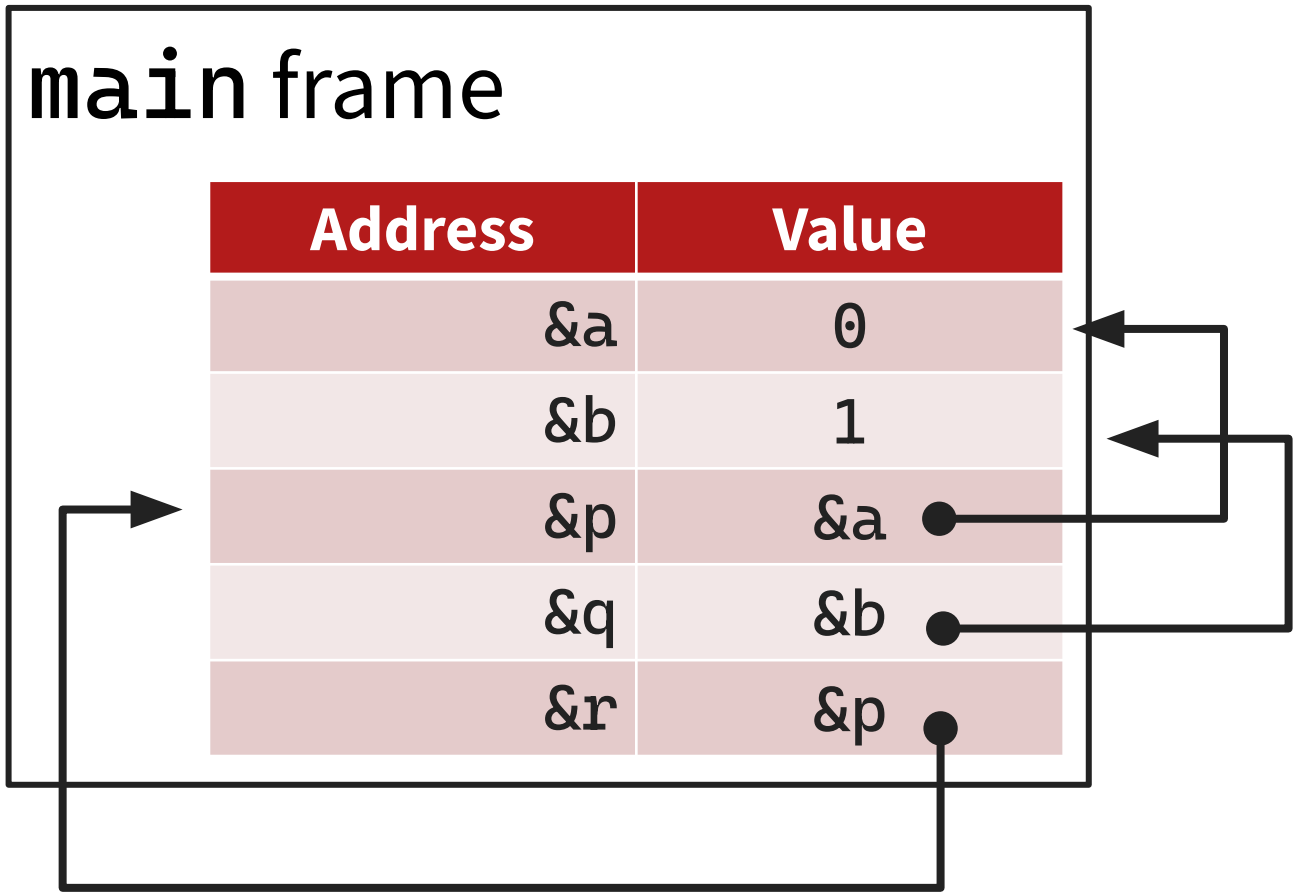
`main` frame

Address	Value
&a	0
&b	1
&p	&a
&q	&b
&r	



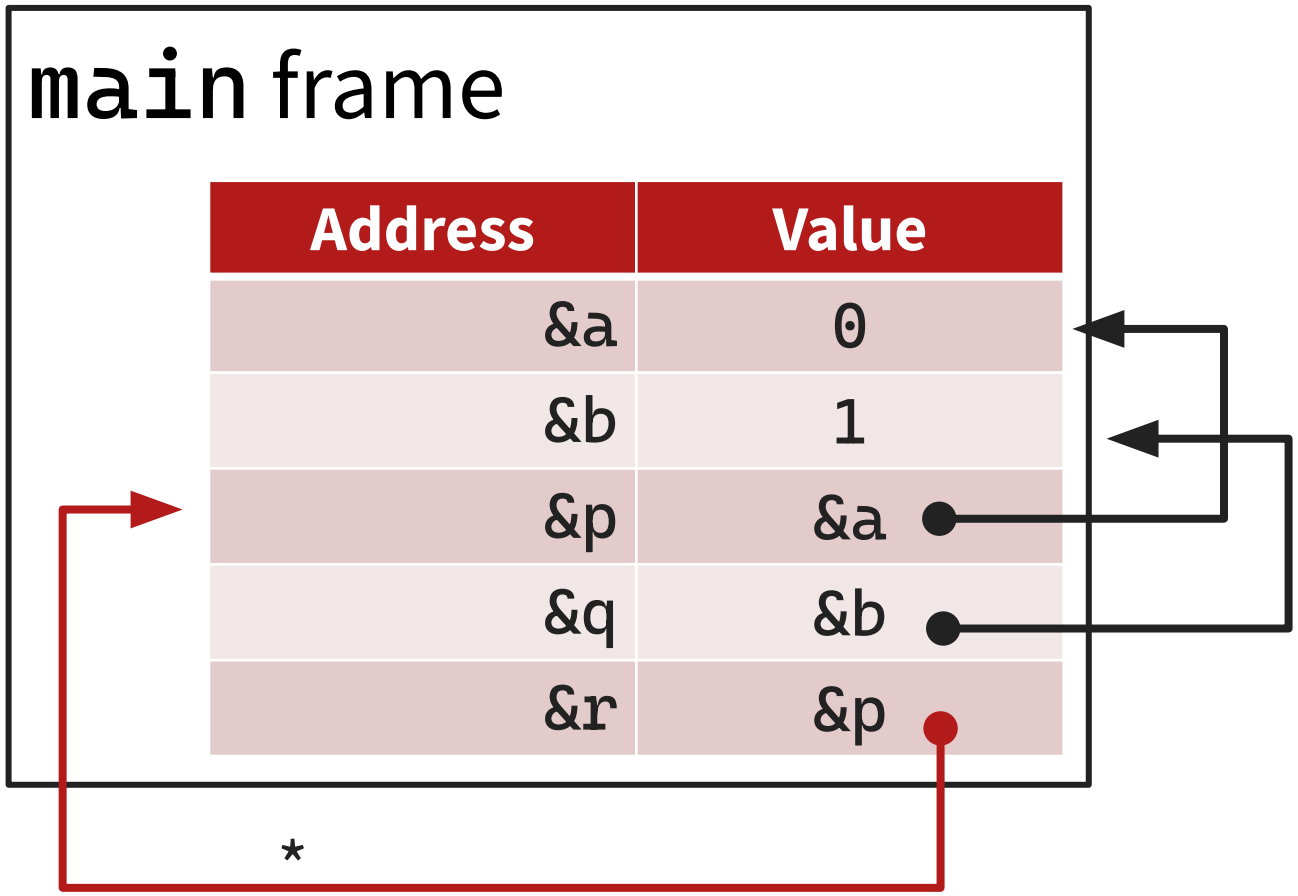
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



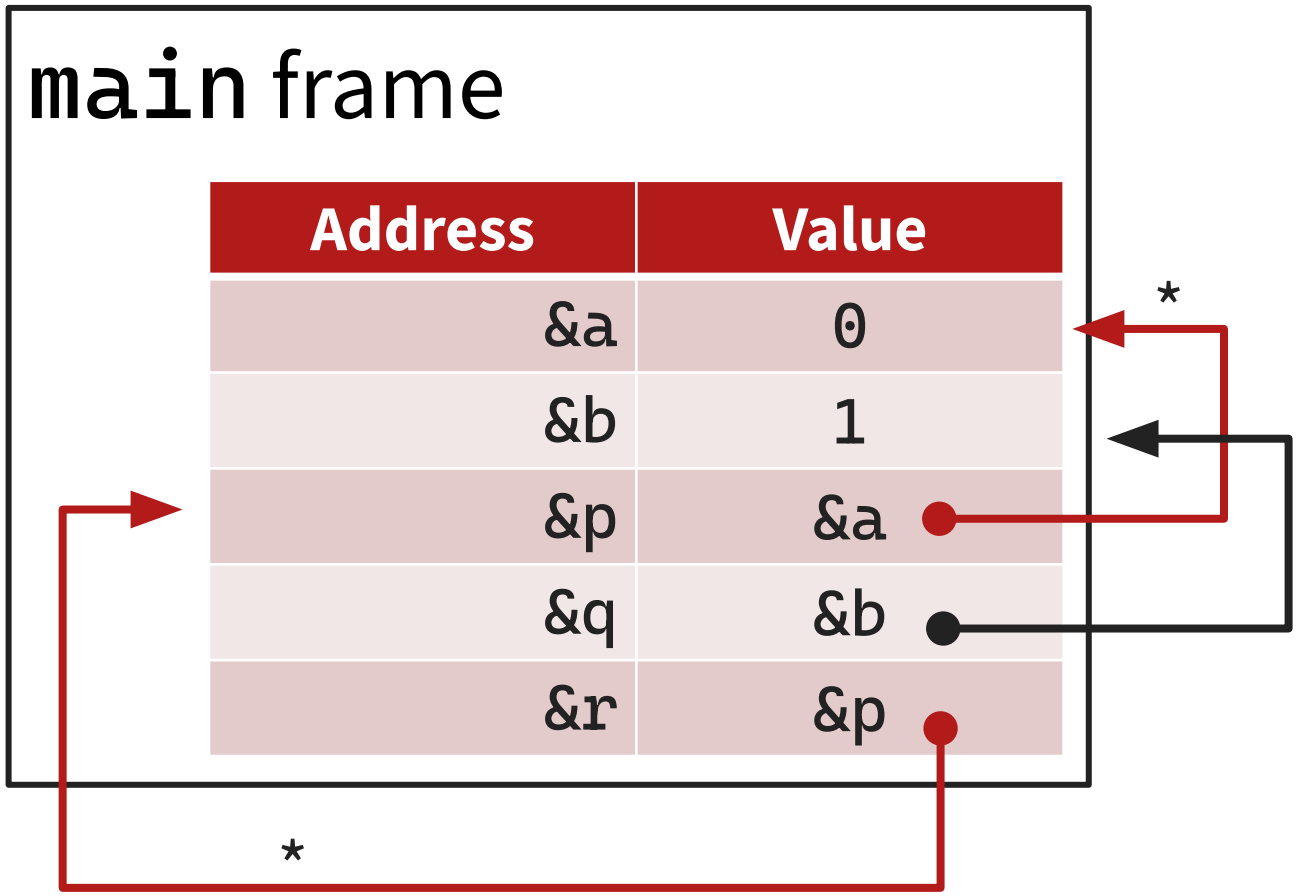
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



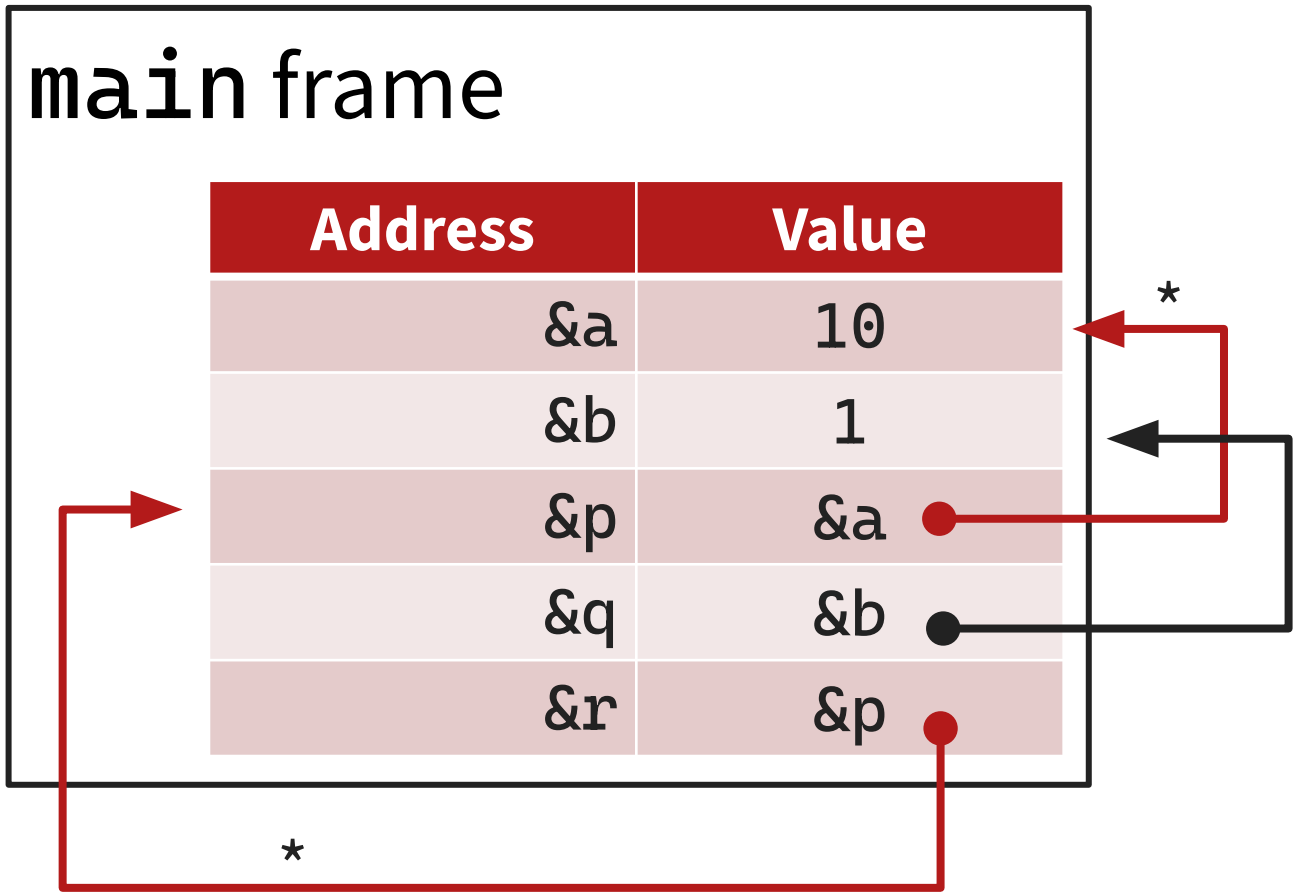
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



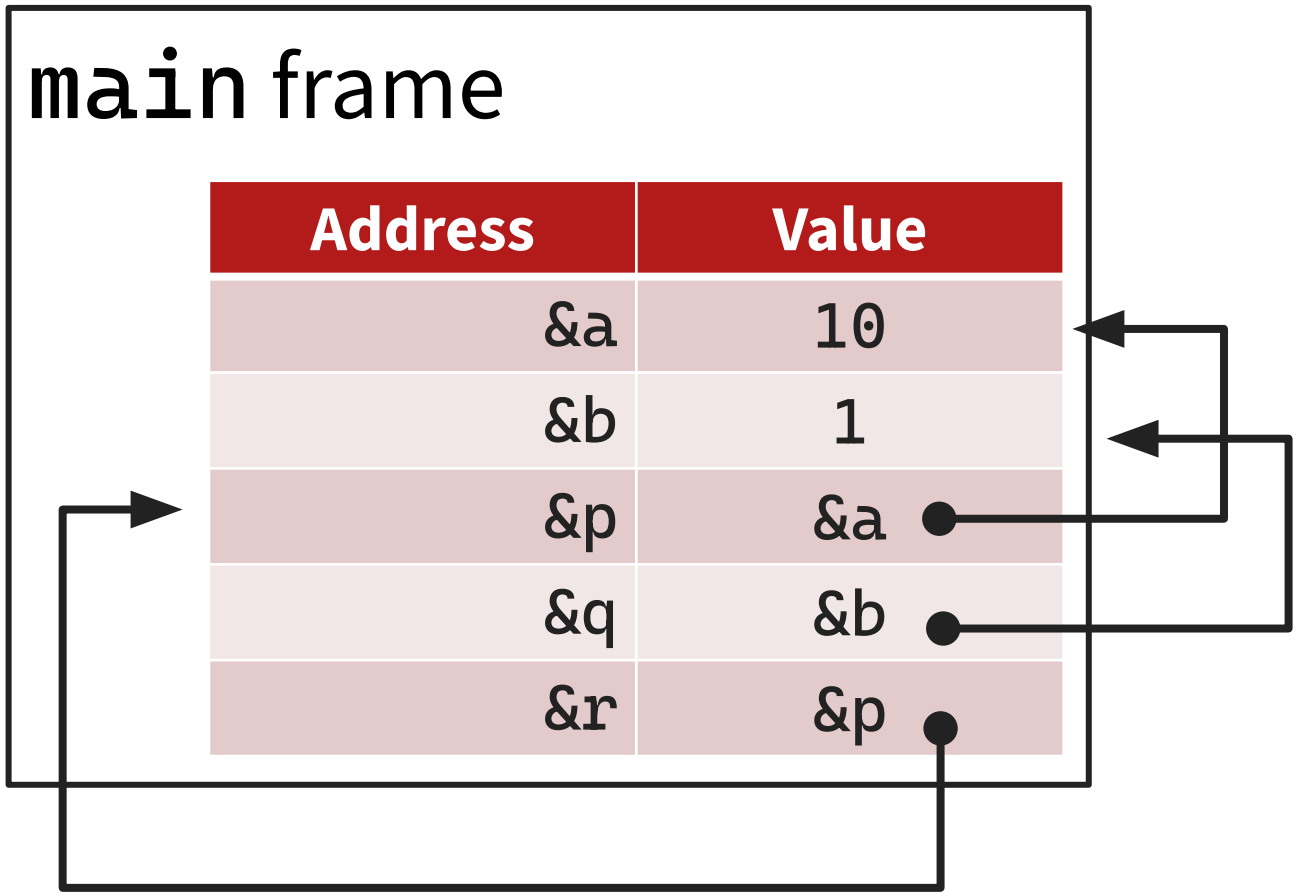
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



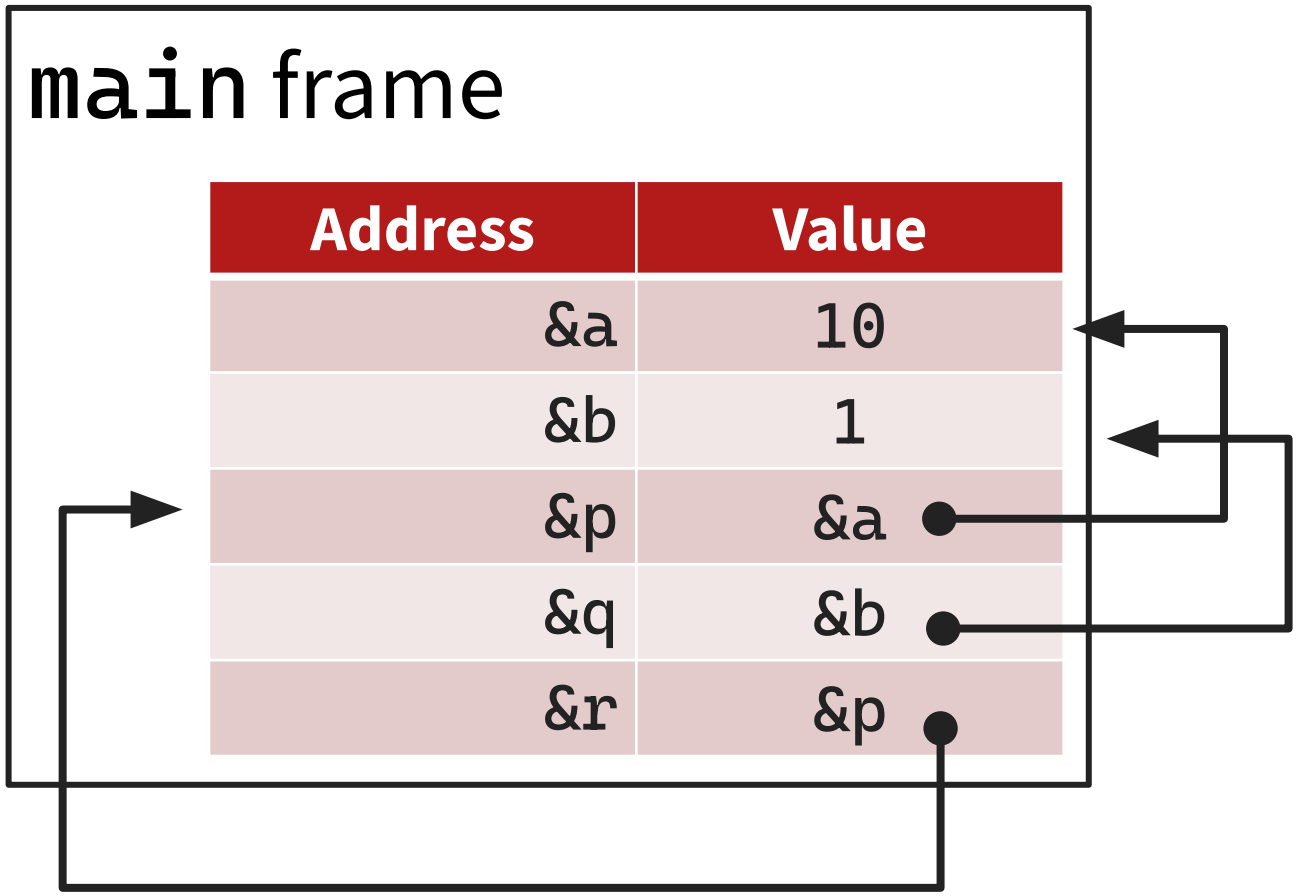
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



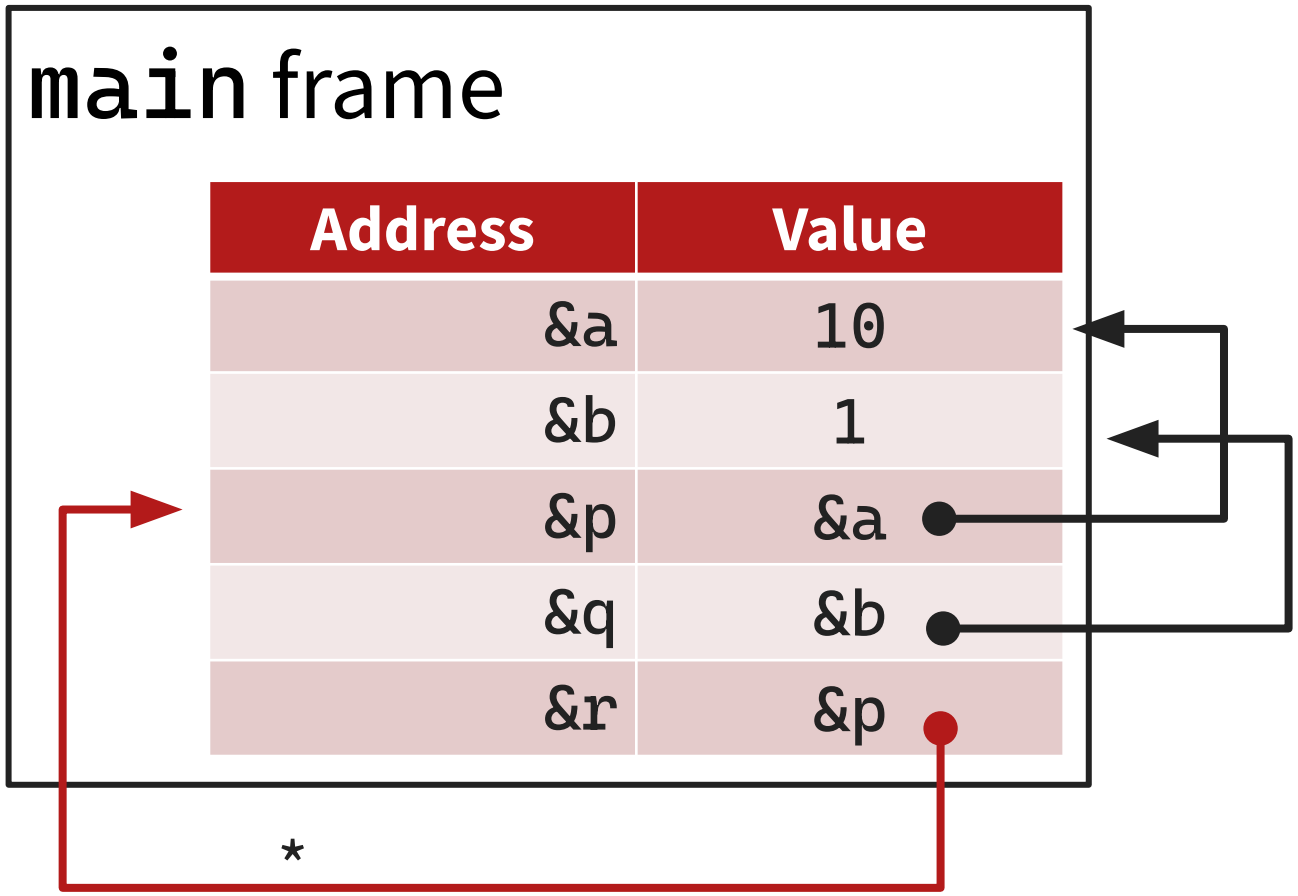
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



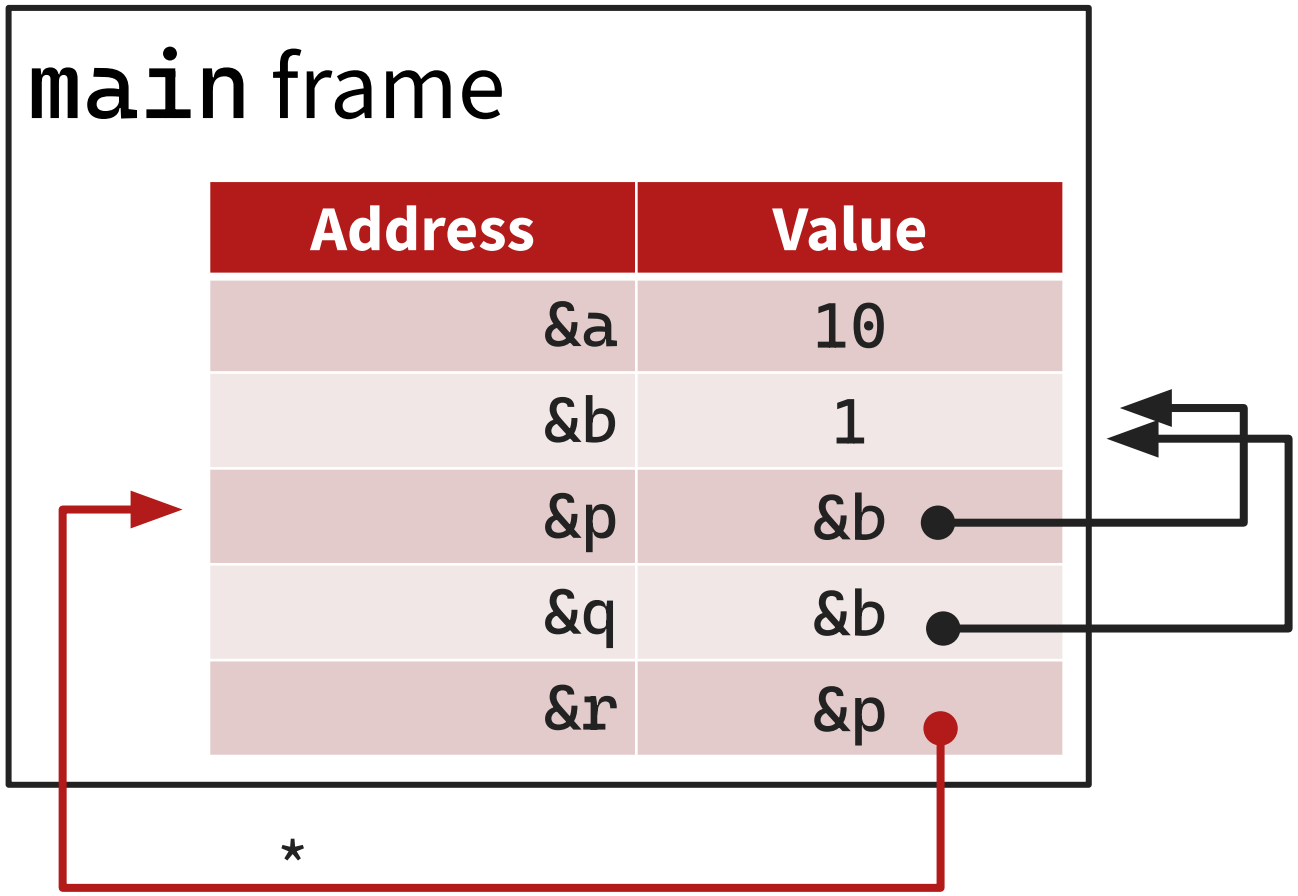
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



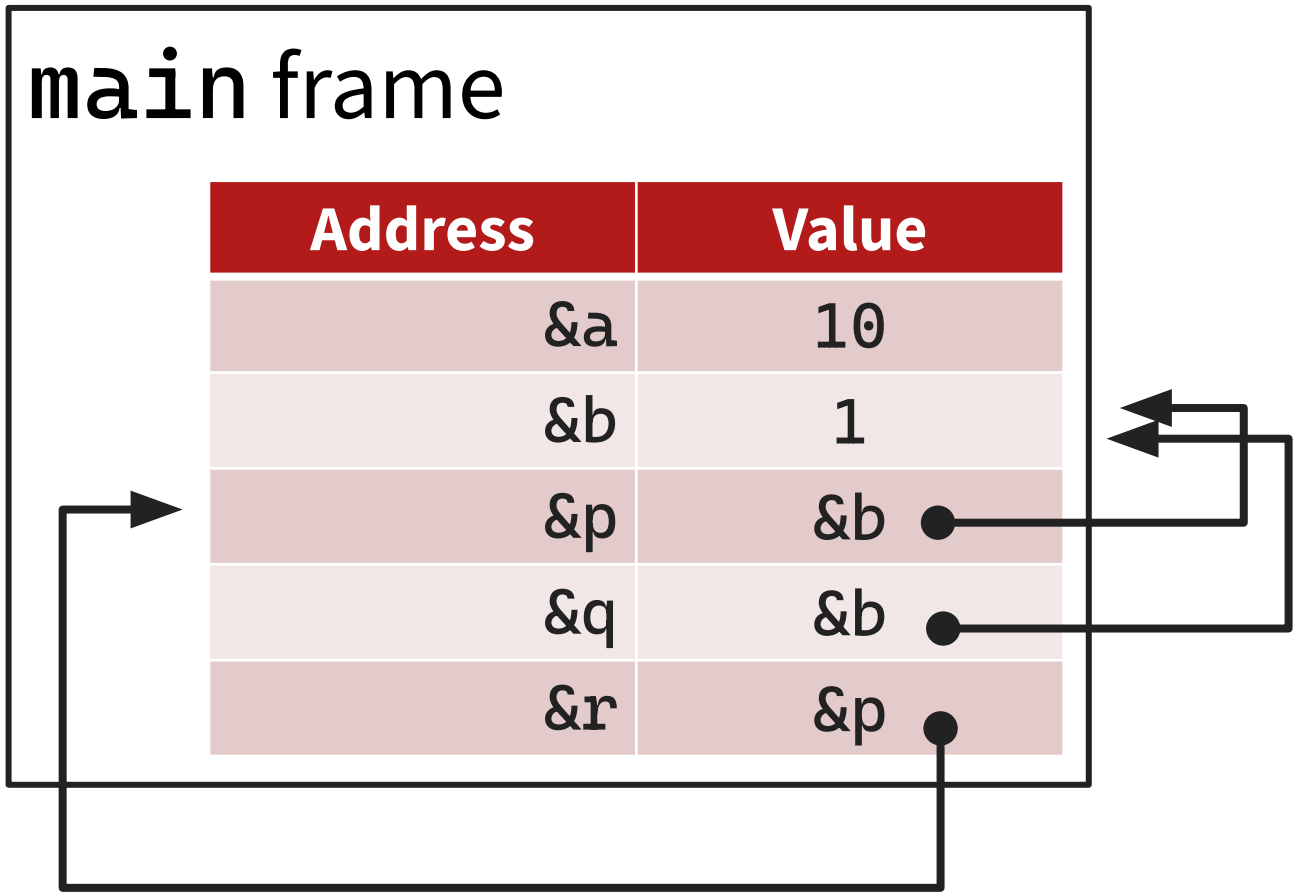
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



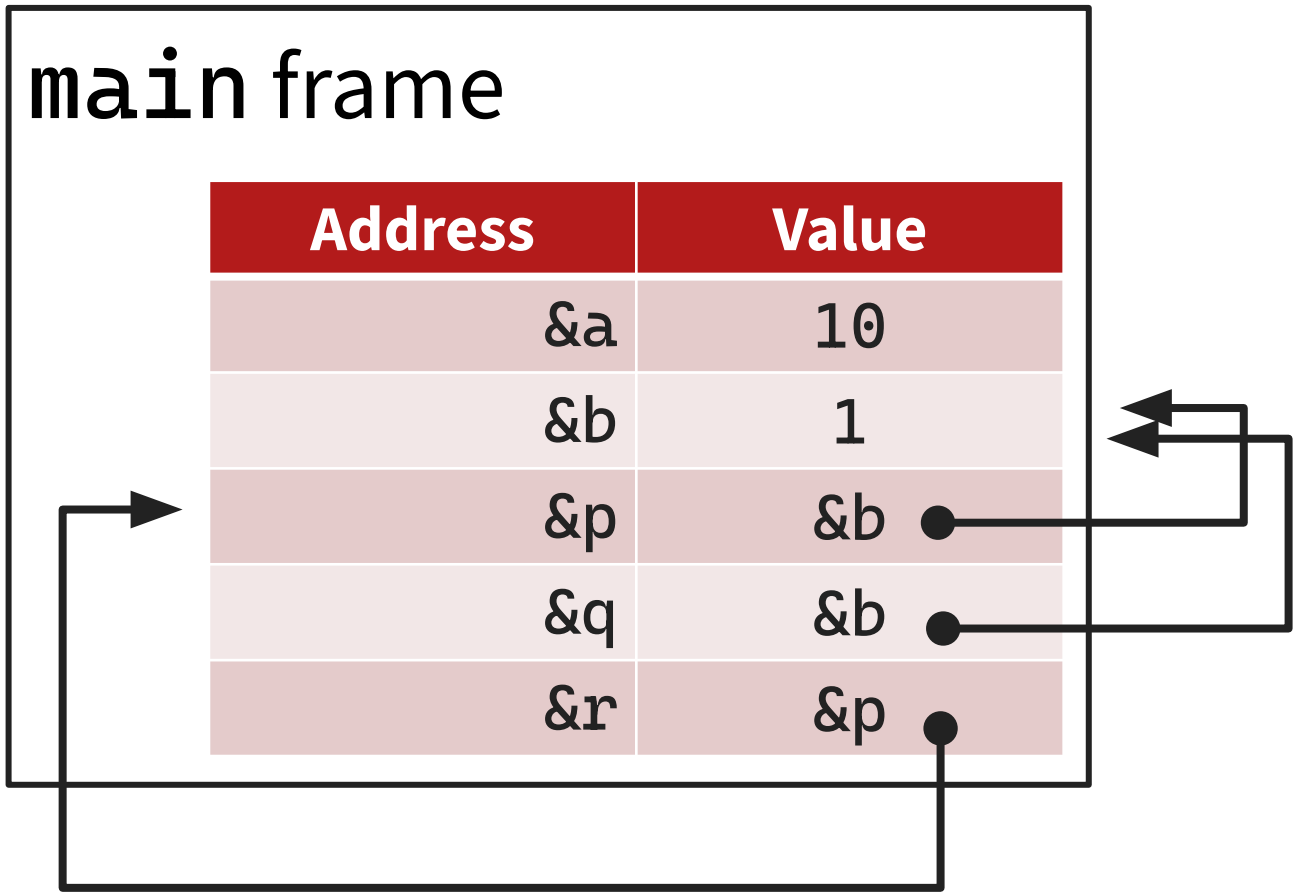
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



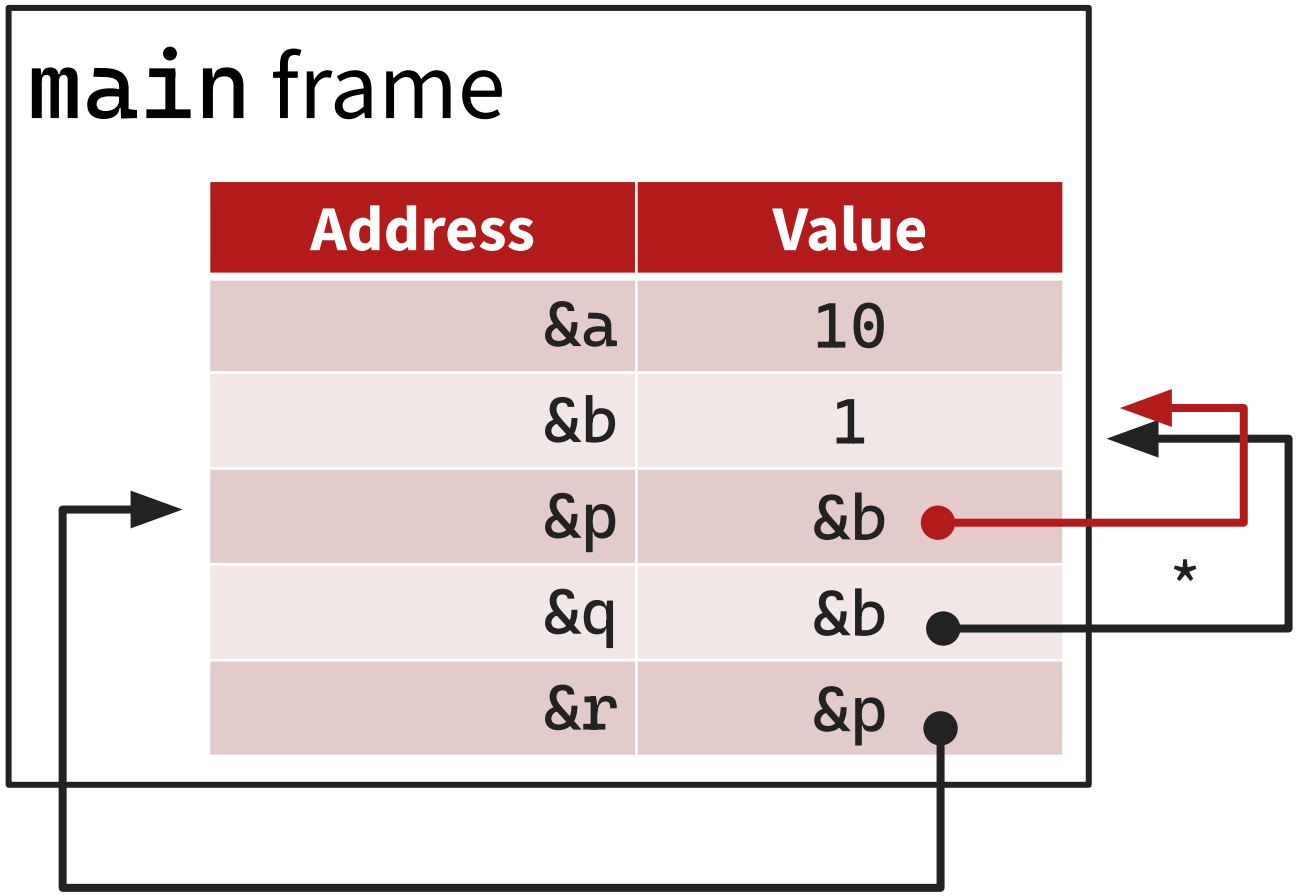
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



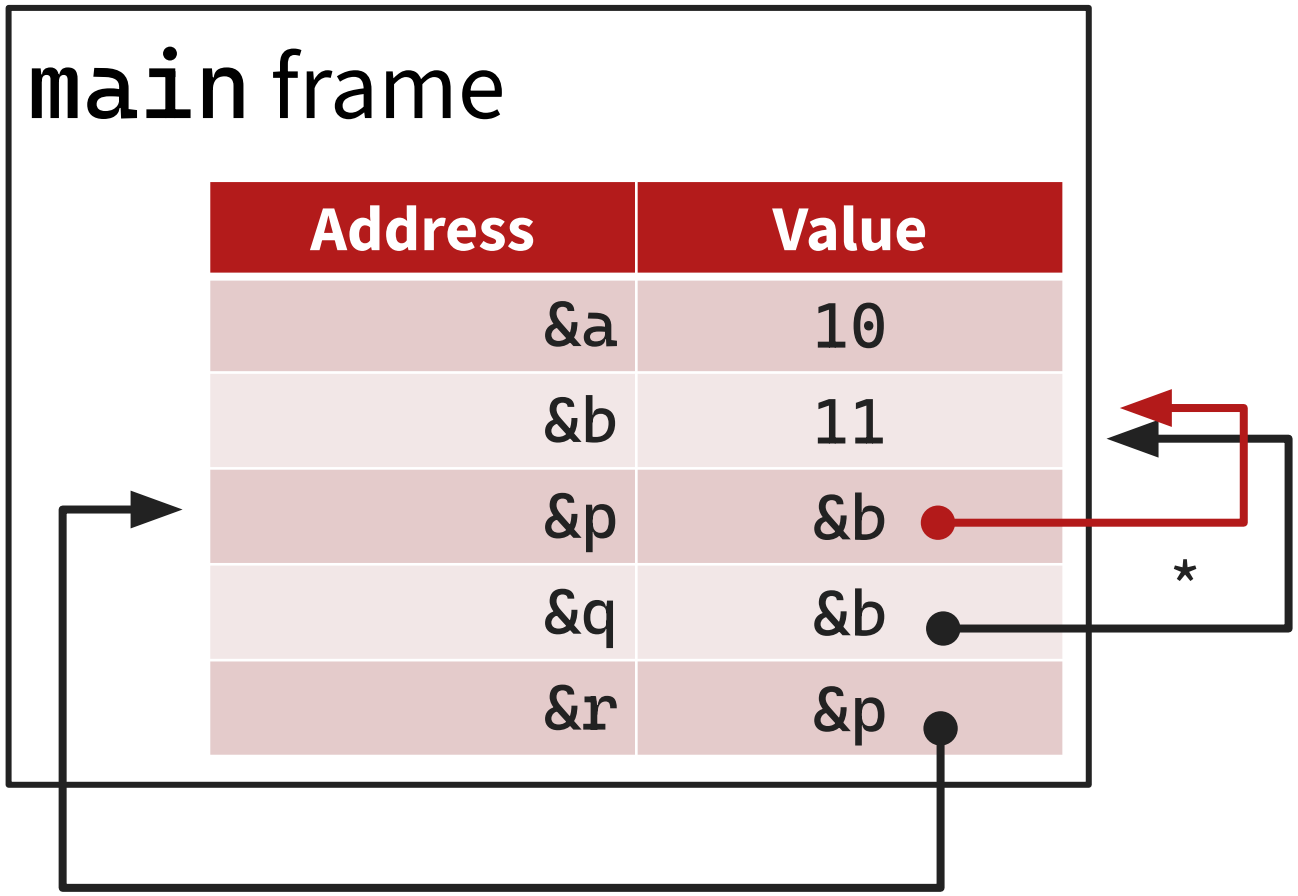
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



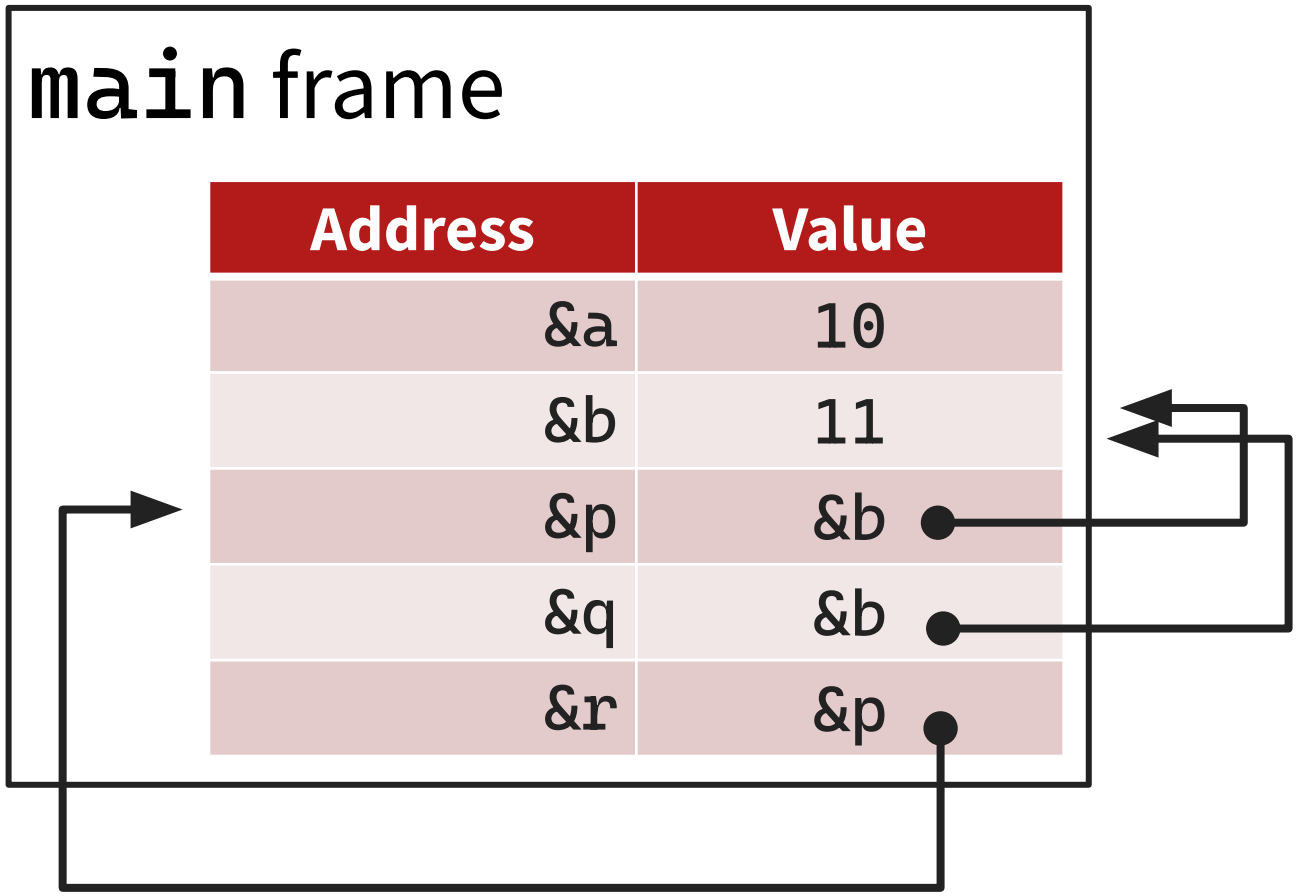
Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



Pointer Diagrams

```
 1  int main() {
 2    uint8_t a = 0;
 3    uint8_t b = 1;
 4    uint8_t *p = &a;
 5    uint8_t *q = &b;
 6    uint8_t **r = &p;
 7    **r = 10;
 8    *r = q;
 9    *p = 11;
10    return 0;
11  }
```



Strings

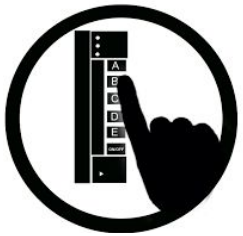
Strings are Null-Terminated Character Arrays

- Recall that we told you a string has type `char*` in C
  - Strings are arrays of `char` values
  - A `char` is generally 1-byte (8-bits)
- Strings keep track of length by ending with a *null character* (`'\0'`)
  - All strings *should* end with a *null character*
- **Example:**
  - `"CS3410" = { 'C', 'S', '3', '4', '1', '0', '\0' }`
  - "CS3410" takes up 7 bytes in memory, not 6!

Select all true statements

- strlen(...) in C computes in O(n)
- strlen(...) in C computes in O(1)
- .length() in Java computes in O(n)
- .lenght() in Java computes in O(1)
- C and Java strings take about the same amount of memory
- C strings are more compact than Java strings

PollEv.com/cs3410



75

Demo: Strings

```
 1  void print_line(char *s) {
 2    for (int i = 0; s[i] ≠ '\0'; ++i)
 3    {
 4      fputc(s[i], stdout);
 5    }
 6    fputc('\n', stdout);
 7  }
 8
 9  int main() {
10    char message[7] = {'H', 'e', 'l', 'l', 'o', '!', '\0'};
11    print_line(message);
12    return 0;
13  }
```

Next Time

- unified memory model
- pointers as addresses
- storage duration
- stack and heap