ECS 36A, April 25, 2024

Announcements

- Midterm has been moved to Tuesday, May 7
  - It was scheduled for Thursday, May 2
- Midterm study guide, sample midterm are on Canvas
  - Sample midterm is shorter than the real one will be
  - I will post answers to it on Monday, April 29
- Tutoring is available from the CS Tutoring Club
  - See the announcement on Canvas

Recursion

- Sometimes it is easier to express a problem in terms of itself, but smaller
- Example: n! defined as $n! = 1 \times 2 \times \ldots \times n$ if $n > 0$ and $0! = 1$.
- Alternate way: $n! = n \times (n-1)!$; $0! = 1$;
- Another example: Fibonacci numbers; each number is the sum of the two preceding Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …

Structure of a Recursive Function

- Base case: this says when to stop the recursion
- Recursive case: this states the recursion part
- *Important: the recursive case must reduce the number of times the function will recurse*
  - In other words, it has to get closer to the base case

```
 1:int nfact(int n)
 2:{
 3:    int j;
 4:    int prod = 1;
 5:
 6:    /* special case: 0! = 1 */
 7:    if (n == 0) return(1);
 8:
 9:    /* ordinary case: loop */
10:    for(j = 1; j <= n; j++)
11:           prod *= j;
12:
13:    /* done */
14:    return(prod);
15:}
```

```
16:
17:int main(void)
18:{
19:    int n;
20:
21:    n = nfact(4);
22:    printf("4! is %d\n", n);
23:    return(0);
24:}
```

```
 1:int nfact(int n)
 2:{
 3:    int x;
 4:
 5:    /* base case: check for 0 */
 6:    if (n == 0) return(1);
 7:
 8:    /* recurse */
 9:    x = nfact(n-1);
10:
11:    /* done! */
12:    return(n * x);
13:}
```

```
14:
15:int main(void)
16:{
17:    int n;
18:
19:    n = nfact(4);
20:    printf("4! is %d\n", n);
21:    return(0);
22:}
```

Initial call to nfact: nfact(n ← 4)

```
 1:int nfact(int n)
 2:{
 3:    int x;
 4:
 5:    /* base case: check for 0 */
 6:    if (n == 0) return(1);
 7:
 8:    /* recurse */
 9:    x = nfact(n-1);
10:
11:    /* done! */
12:    return(n * x);
13:}
```

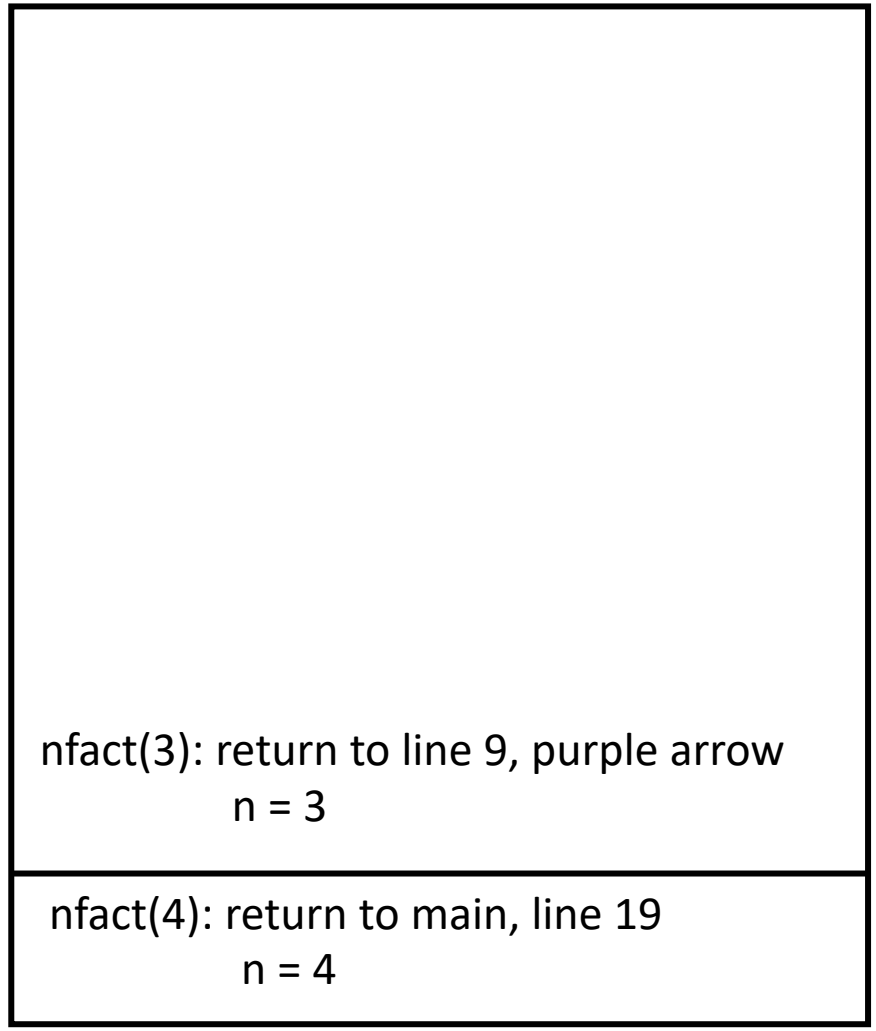
nfact(4): return to main, line 19
n = 4

nfact(n ← 4):

6: condition false, so skip

9: call nfact(4–1), or nfact(3)

```
 1:int nfact(int n)
 2:{
 3:    int x;
 4:
 5:    /* base case: check for 0 */
 6:    if (n == 0) return(1);
 7:
 8:    /* recurse */
 9:    x =    nfact(n-1);
10:
11:    /* done! */
12:    return(n * x);
13:}
```

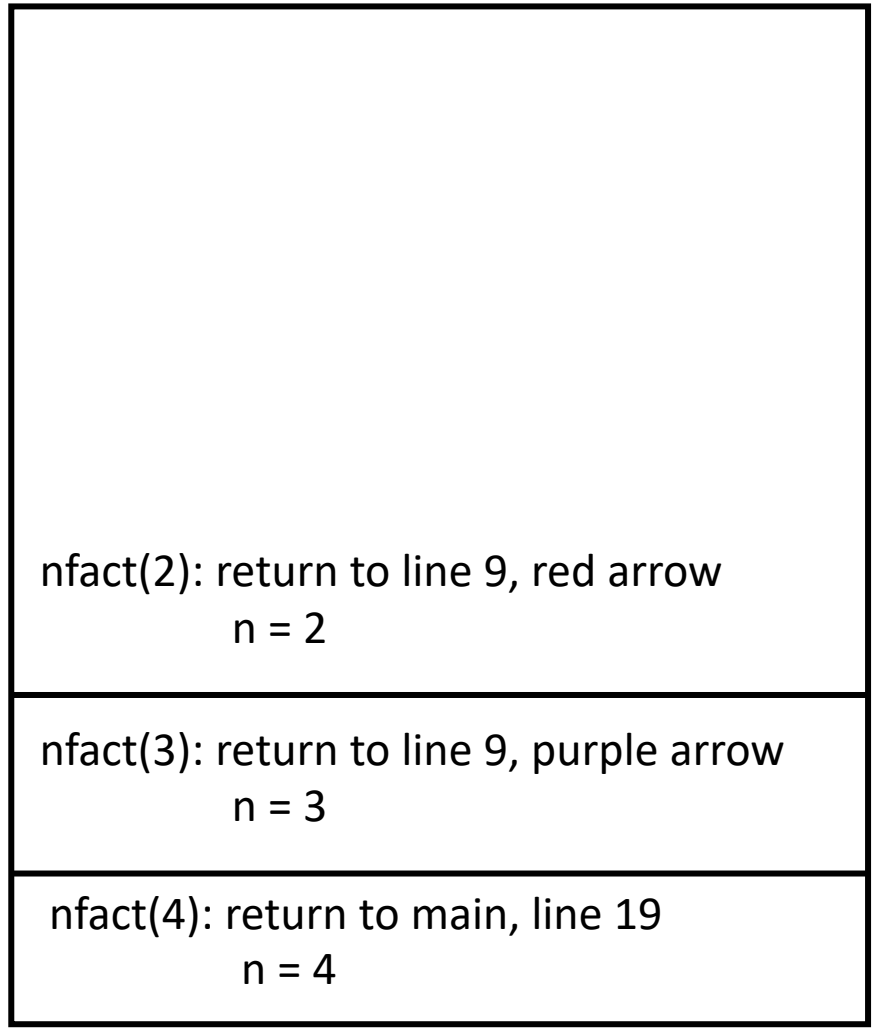


nfact(n ← 3):

6: condition false, so skip

9: call nfact(3–1), or nfact(2)

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x =     nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```



nfact(n ← 2):

6: condition false, so skip

9: call nfact(2–1), or nfact(1)

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x = nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```

nfact(1): return to line 9, blue arrow n = 1
nfact(2): return to line 9, red arrow n = 2
nfact(3): return to line 9, purple arrow n = 3
nfact(4): return to main, line 19 n = 4

nfact(n ← 1):

6: condition false, so skip

9: call nfact(1–1), or nfact(0)

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x = nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```

nfact(0): return to line 9, green arrow n = 0
nfact(1): return to line 9, blue arrow n = 1
nfact(2): return to line 9, red arrow n = 2
nfact(3): return to line 9, purple arrow n = 3
nfact(4): return to main, line 19 n = 4

nfact(n ← 0):

6: condition true, so return 1

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x = nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```

nfact(0): return to line 9, green arrow n = 0; return 1
nfact(1): return to line 9, blue arrow n = 1; nfact(0) = 1
nfact(2): return to line 9, red arrow n = 2
nfact(3): return to line 9, purple arrow n = 3
nfact(4): return to main, line 19 n = 4

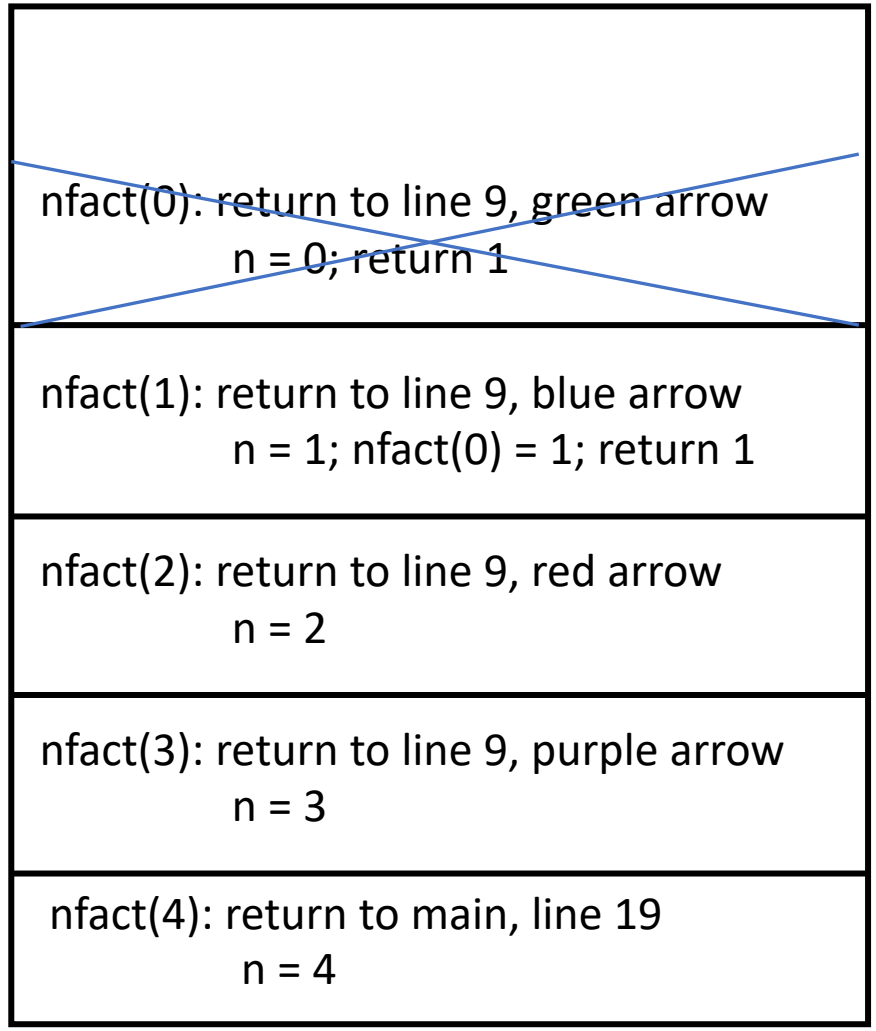
nfact(n ← 1):

6: condition false, so skip

9: call nfact(1–1), or nfact(0); nfact(0) = 1, so x = 1

12: return 1 × 1 = 1

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x = nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```



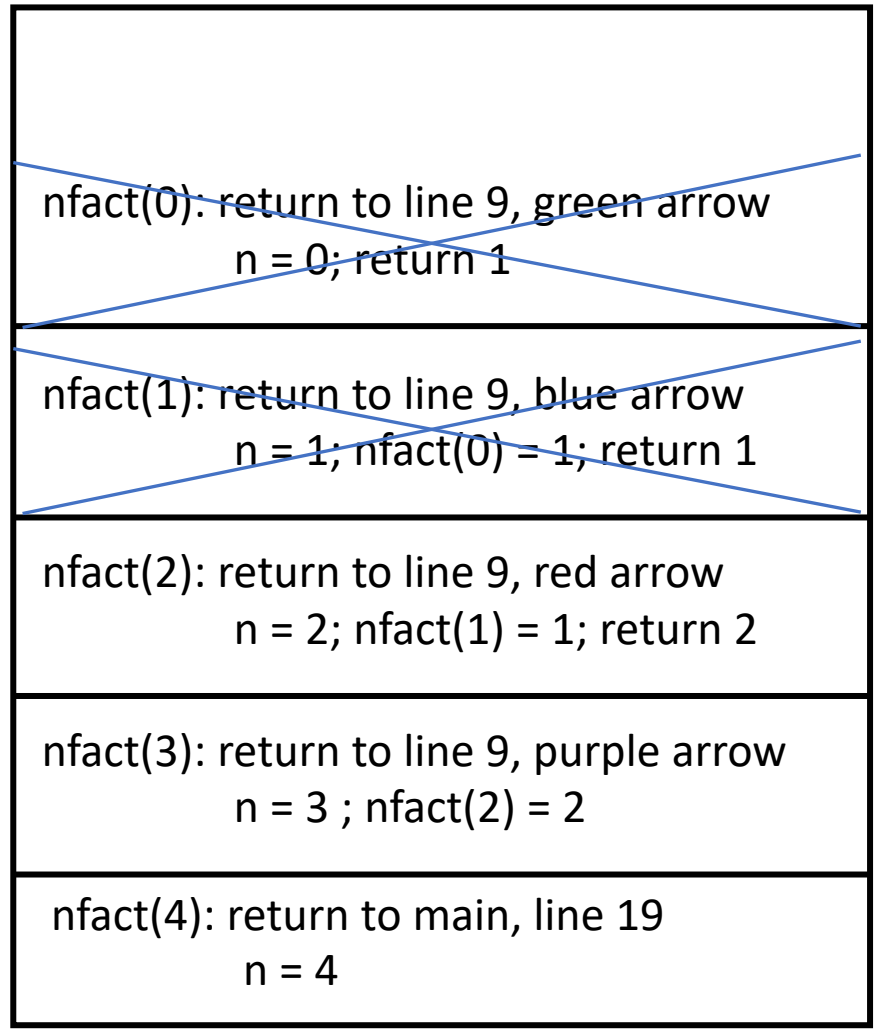
nfact(n ← 2):

6: condition false, so skip

9: call nfact(2–1), or nfact(1); nfact(1) = 1, so x = 1

12: return 2 × 1 = 2

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x =  nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```



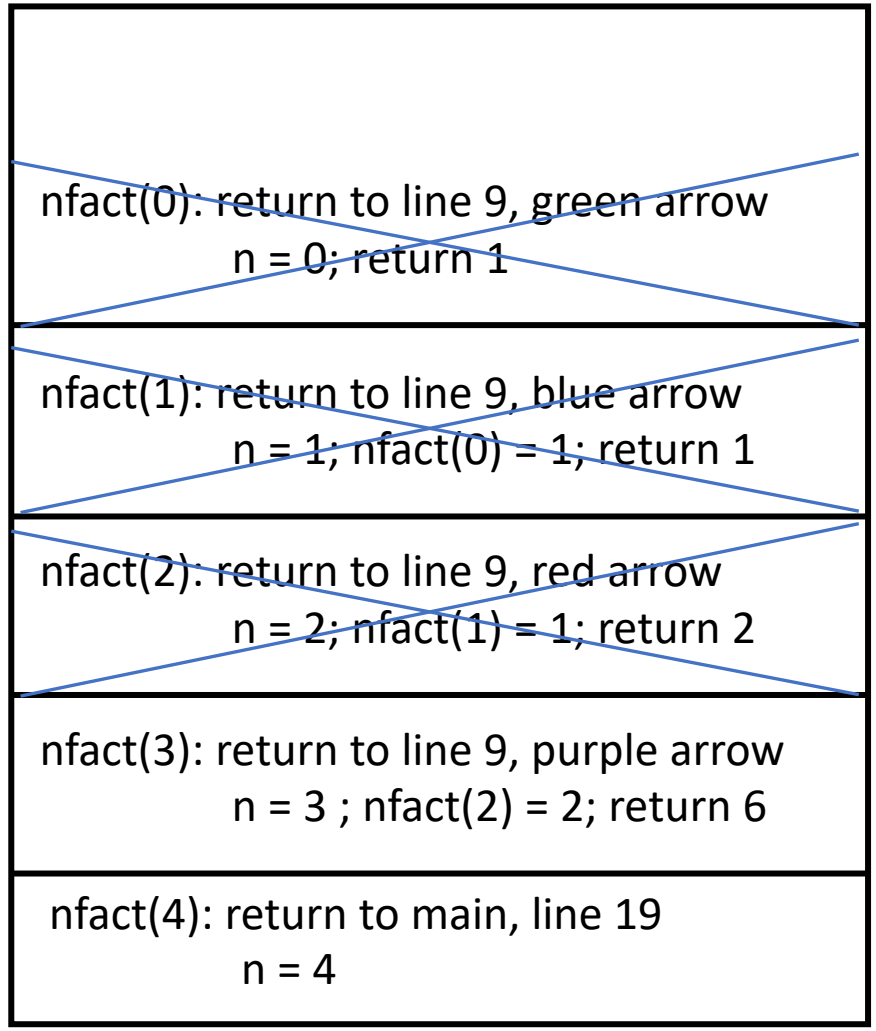
nfact(n ← 3):

6: condition false, so skip

9: call nfact(3–1), or nfact(2); nfact(2) = 2, so x = 2

12: return 3 × 2 = 6

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x =     nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```



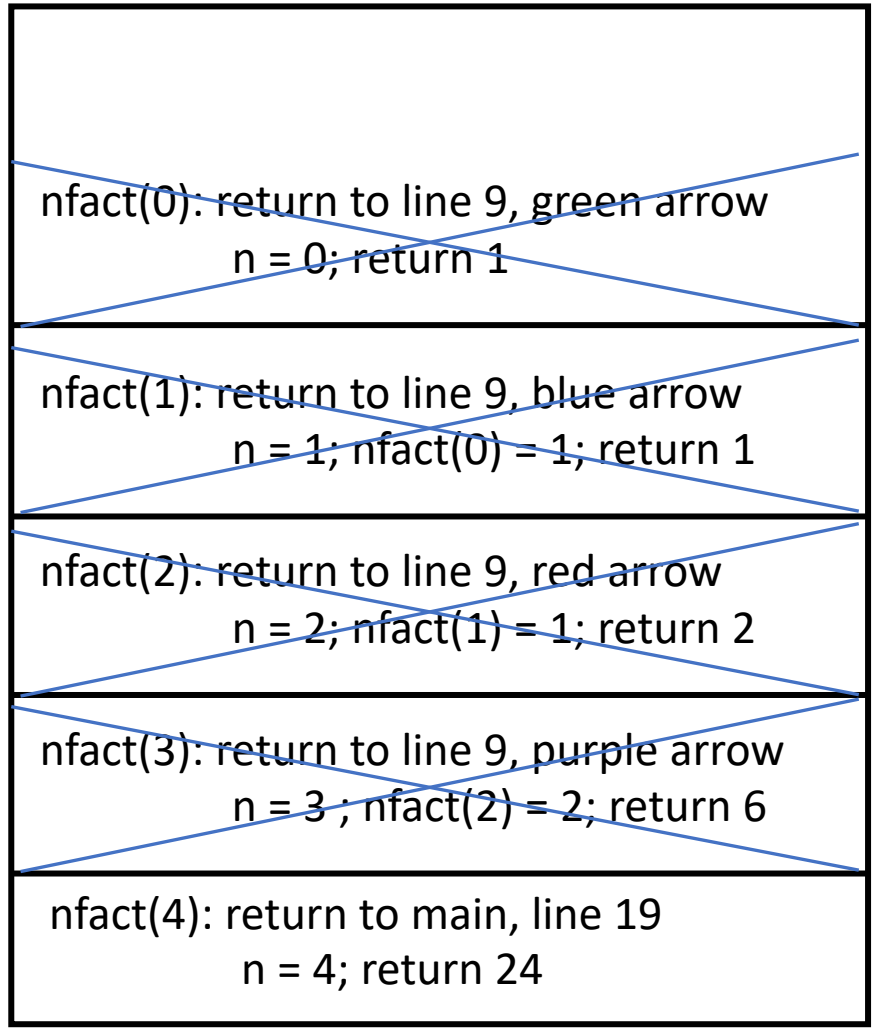
nfact(n ← 4):

6: condition false, so skip

9: call nfact(4–1), or nfact(3); nfact(3) = 6, so x = 6

12: return 4 × 6 = 24

```
 1: int nfact(int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x =    nfact(n-1);
10:
11:     /* done! */
12:     return(n * x);
13: }
```



```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:          return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:          n[strlen(n)-1] = '\0';
10:          return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```

```
14:
15:int main(void)
16:{
17:    char buf[1000];
18:    (void) strcpy(buf, "madam")
19:    if (ispal(buf))
20:          printf("Palindrome\n");
21:    else
22:          printf("Not a palindrome\n");
23:    return(0);
24:}
```

Initial call to ispal: ispal(n ← “madam”)

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```

ispal(“madam”): return to main, line 19
n = “madam”

ispal(n ← “madam”):
4: condition false, so skip
9: call ispal(“ada”)

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```


ispal(“ada”): return to line 10, purple arrow
n = “ada”

ispal(“madam”): return to main, line 19
n = “madam”

ispal(n ← “ada”):
6: condition false, so skip
9: call ispal(“d”)

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```


ispal(“d”): return to line 10, red arrow
n = “d”

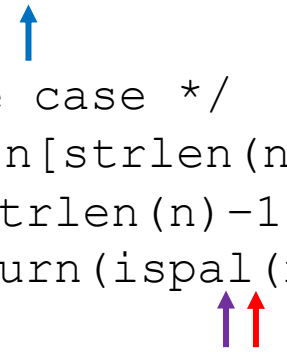
ispal(“ada”): return to line 10, purple arrow
n = “ada”

ispal(“madam”): return to main, line 19
n = “madam”

ispal(n ← “d”):

6: condition true, so return 1

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```

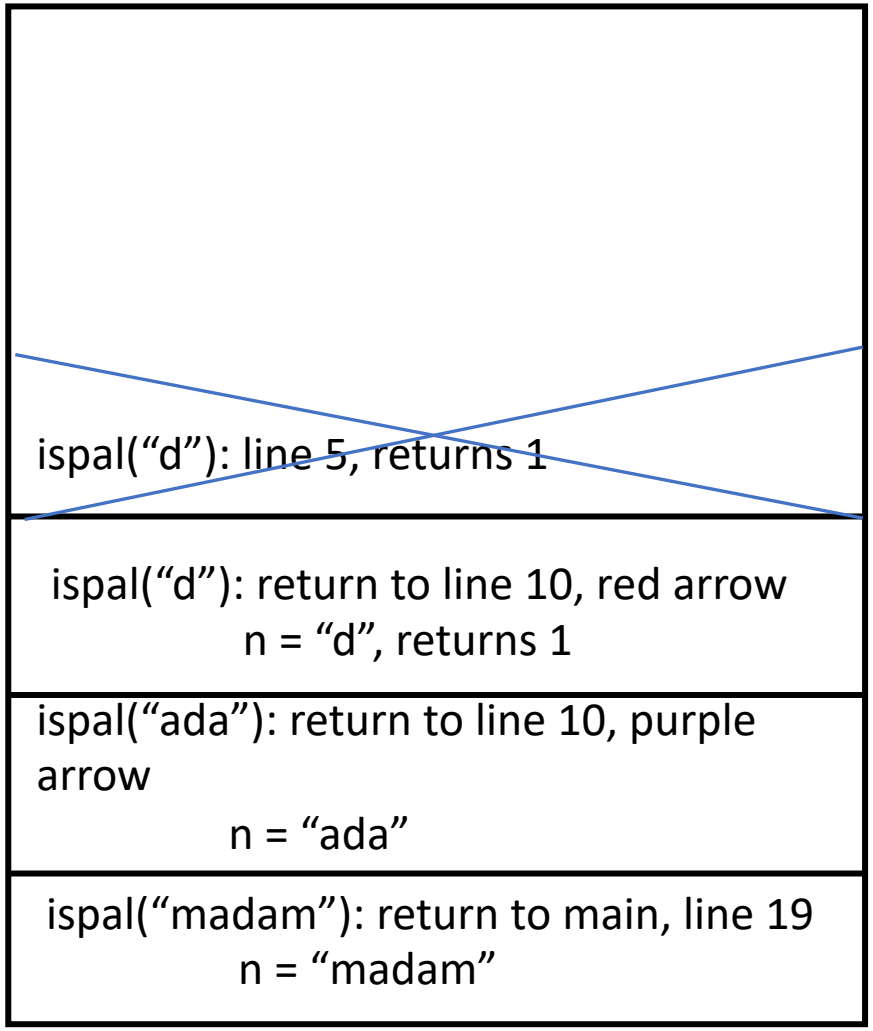


ispal(“d”): do not recurse, blue arrow
ispal(“d”): return to line 10, red arrow n = “d”
ispal(“ada”): return to line 10, purple arrow n = “ada”
ispal(“madam”): return to main, line 19 n = “madam”

ispal(n ← “d”):

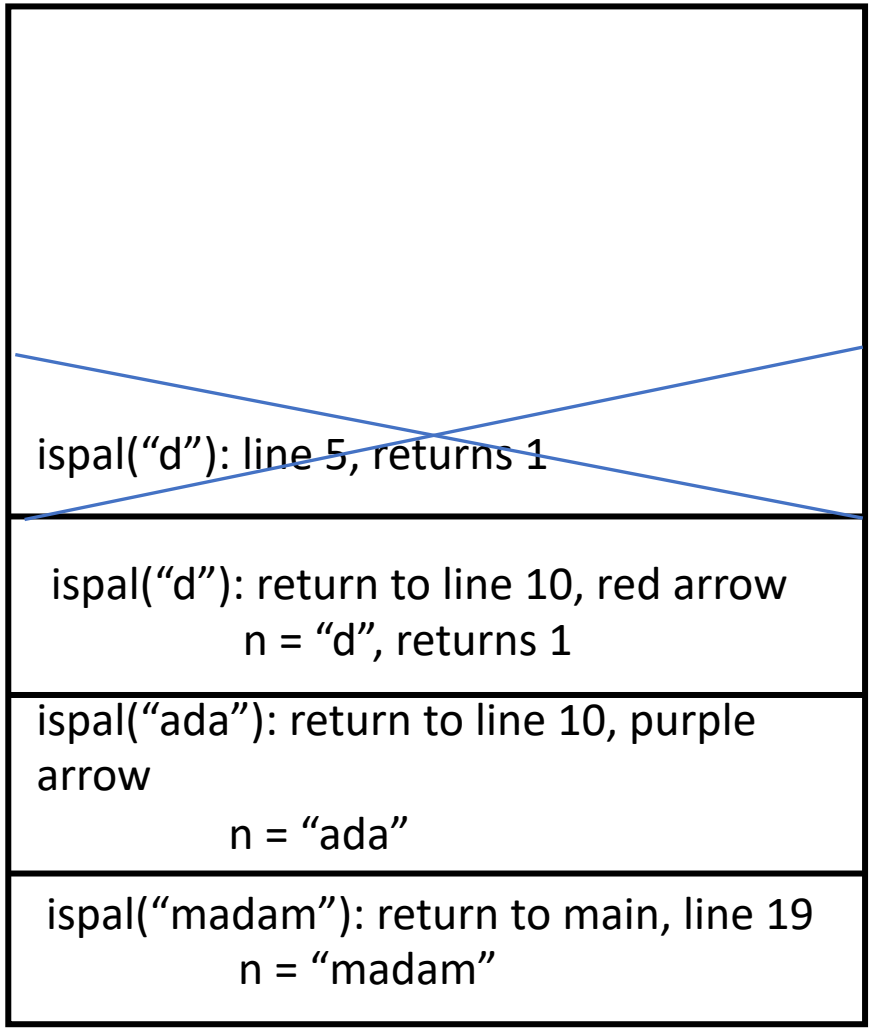
6: condition true, so return 1

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```



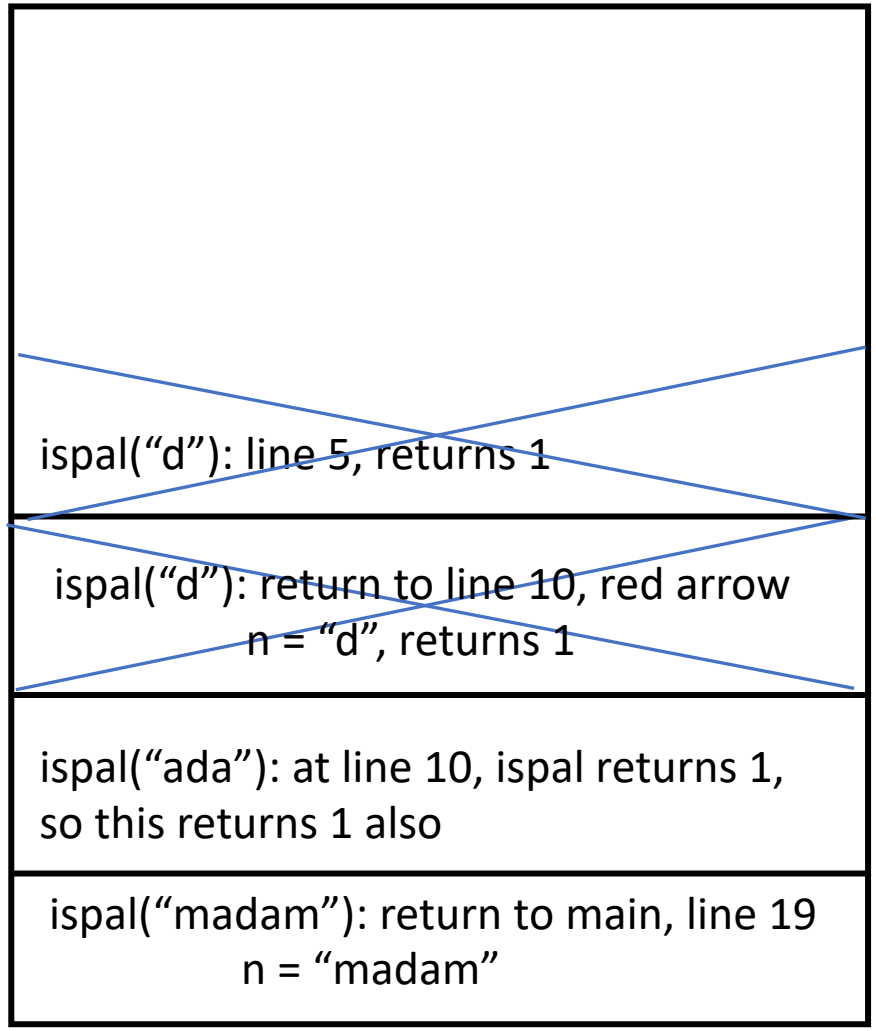
ispal(n ← “d”):
at line 10, return 1

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:          return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:          n[strlen(n)-1] = '\0';
10:          return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```



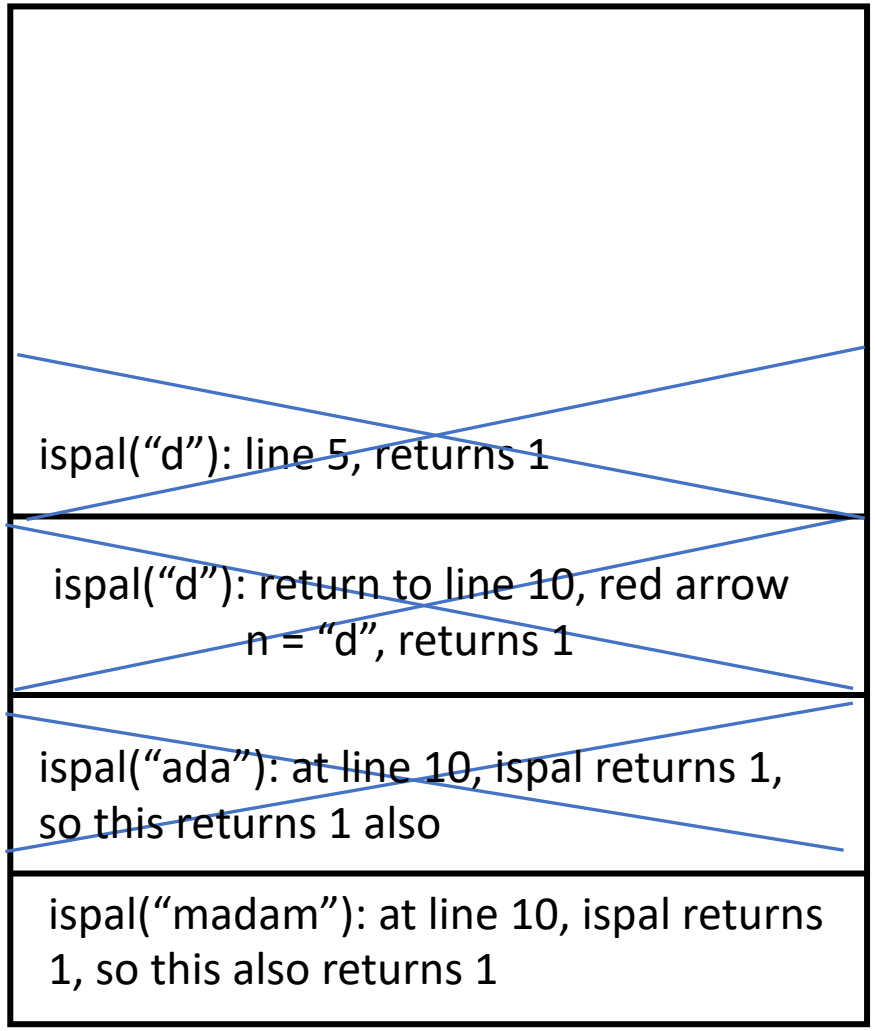
ispal(n ← “ada”):
at line 10, return 1

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```



ispal(n ← “madam”):
at line 10, return 1

```
 1:int ispal(char *n)
 2:{
 3:    /* base case */
 4:    if (!*n || strlen(n) == 1)
 5:           return(1);
 6:
 7:    /* recurse case */
 8:    if (*n == n[strlen(n)-1]){
 9:           n[strlen(n)-1] = '\0';
10:           return(ispal(n+1));
11:    }
12:    else return(0);
13:}
```



Testing for Palindromes

- Approach:
  - If 0 or 1 characters in string, it's a palindrome
  - Otherwise compare the first and last chars; if different, not a palindrome; if the same, see if the middle characters form a palindrome
- Suggested interface:

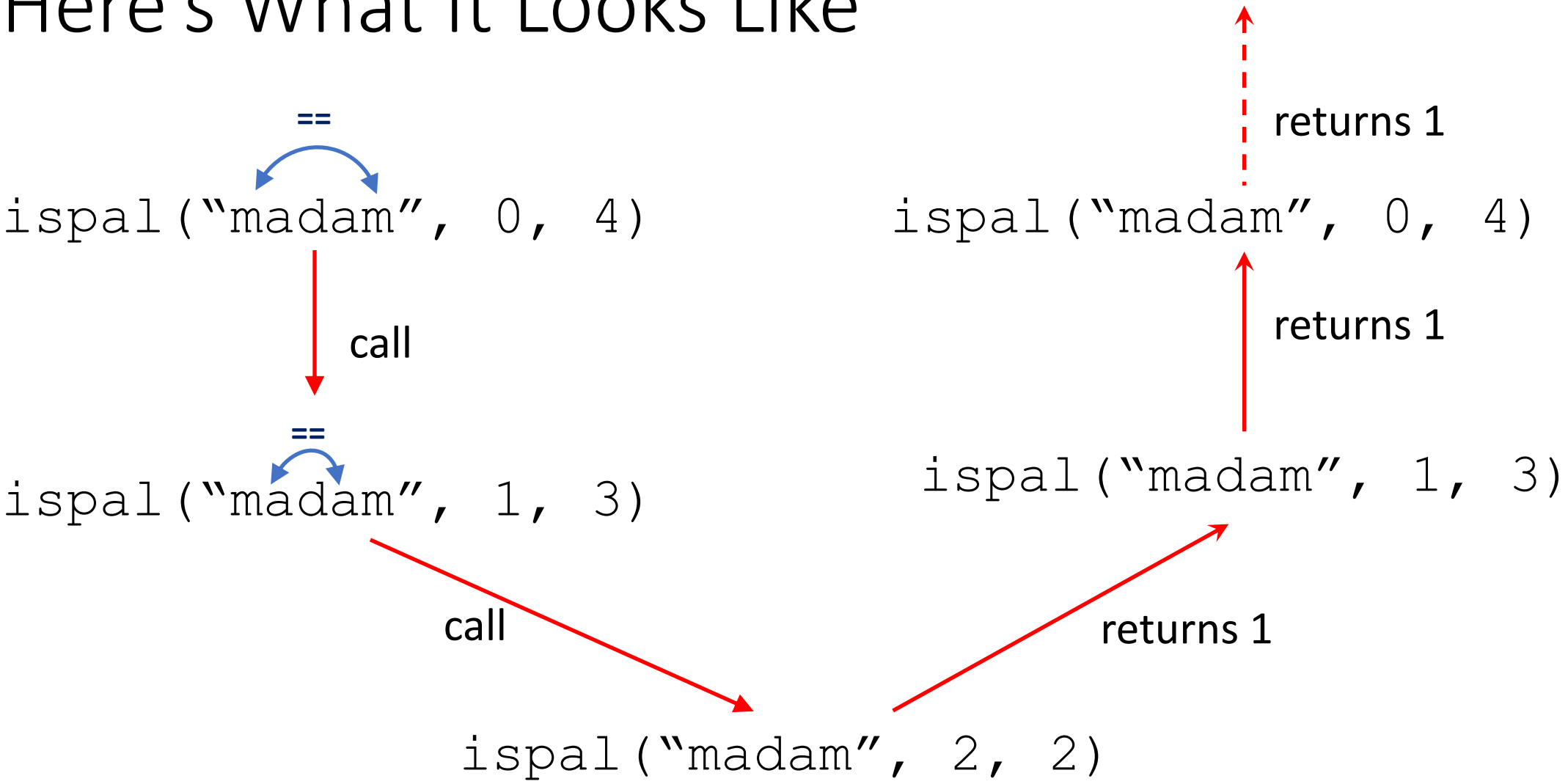
```
int ispal(char *str, int b, int e)
```

The string to be tested

Index of the first char in the string to be tested

Index of the last char in the string to be tested

Here's What It Looks Like

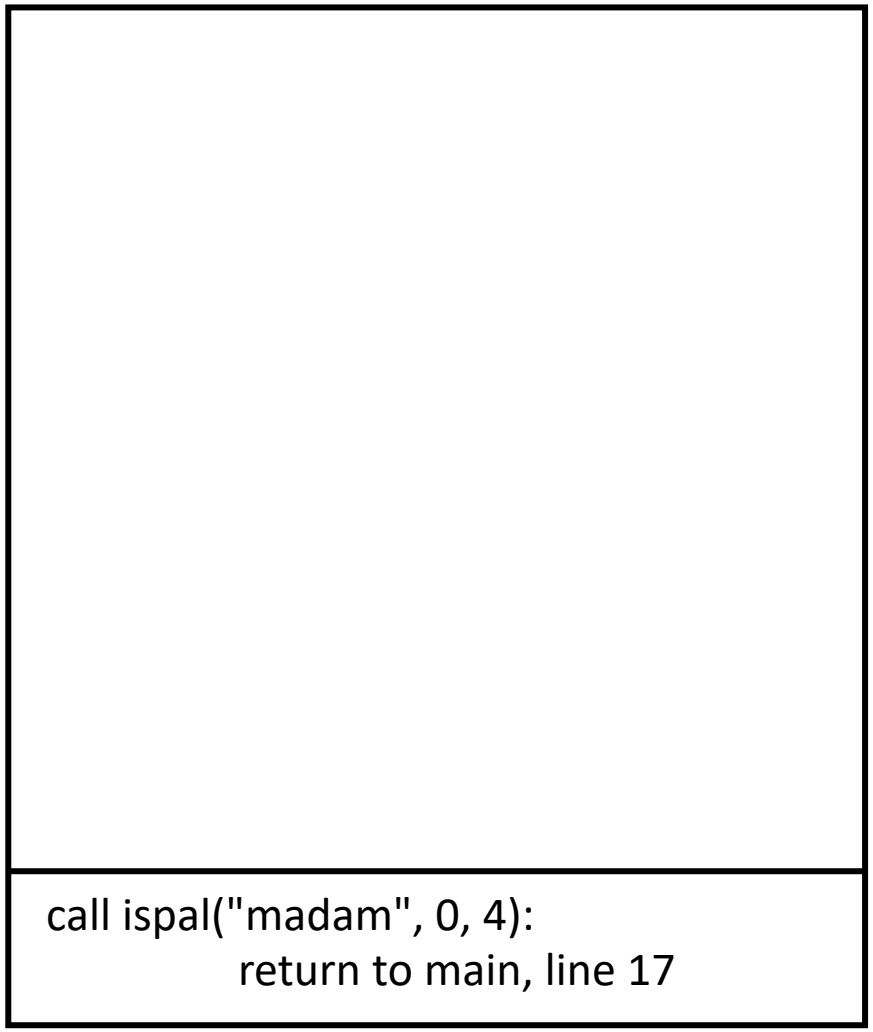


```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:           return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```

```
12:
13:int main(void)
14:{
15:    char buf[1000];
16:    (void) strcpy(buf, "madam")
17:    if (ispal(buf, 0, strlen(buf)-1))
18:           printf("Palindrome\n");
19:    else
20:           printf("Not a palindrome\n");
21:    return(0);
22:}
```

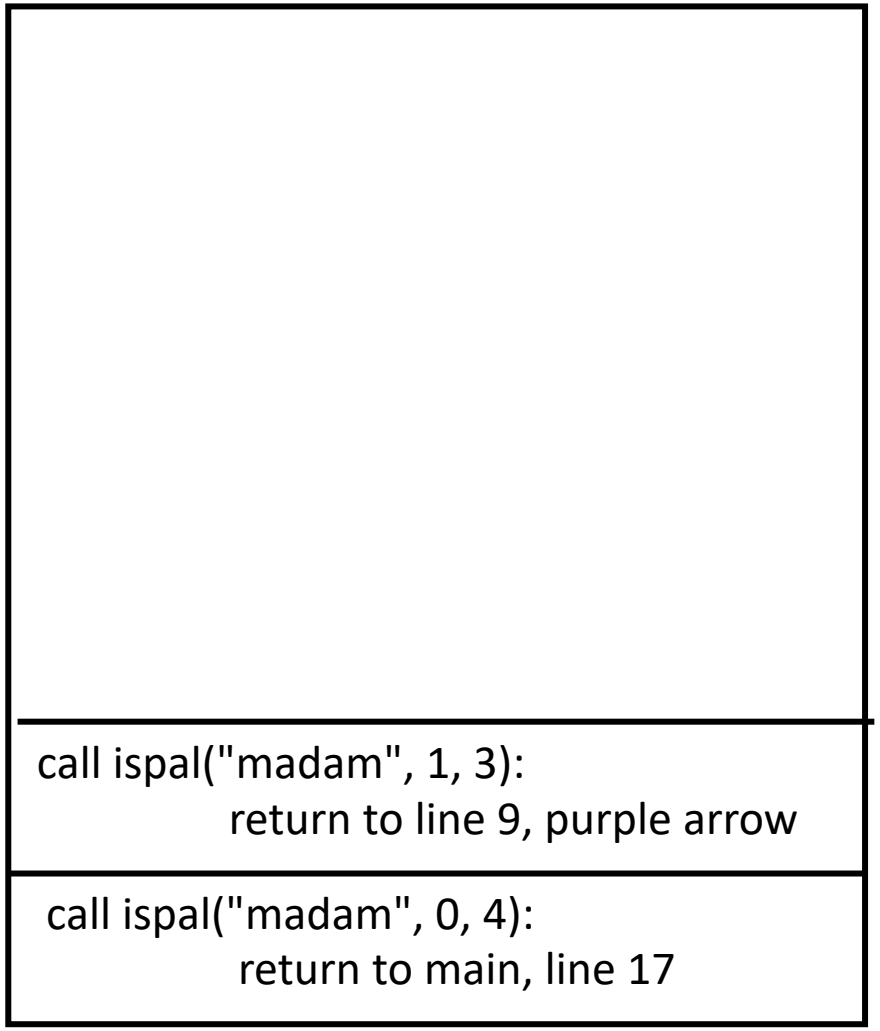
Initial call to ispal: ispal(n ← "madam", b ← 0, e ← 4)

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:           return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```



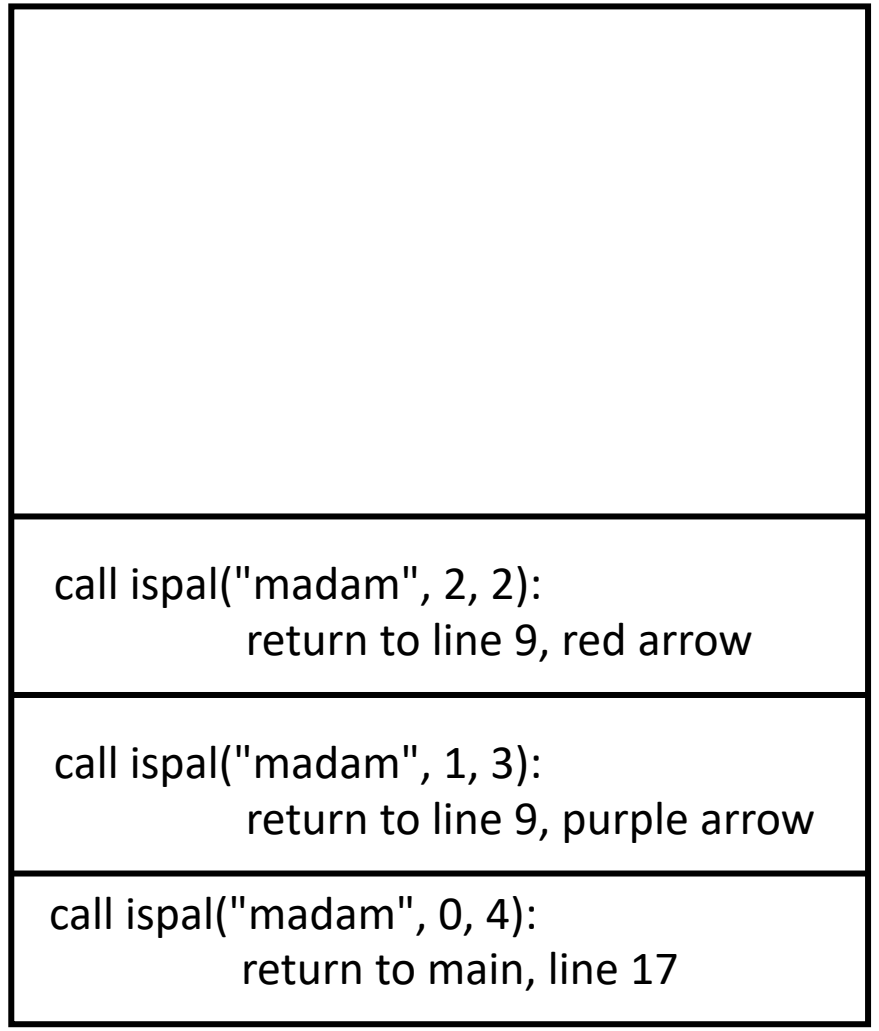
ispal(n ← "madam", b ← 0, e ← 4):
4: condition false, so skip
9: call ispal("madam", 1, 3)

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:           return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```

ispal(n ← "madam", 1, 3):
4: condition false, so skip
9: call ispal("madam", 2, 2)

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:           return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```



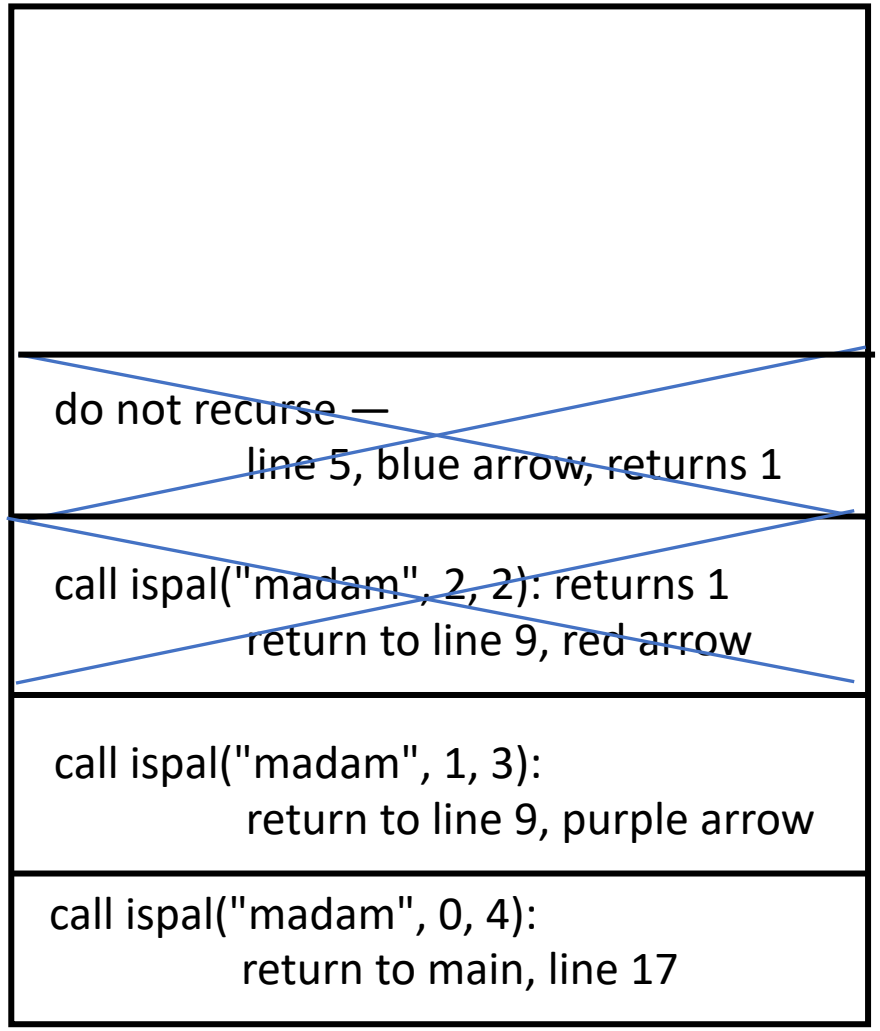
ispal(n ← "madam", 2, 2):
4: condition true, so go to line 5
5: return 1

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:        return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```


do not recurse — line 5, blue arrow, returns 1
call ispal("madam", 2, 2): return to line 9, red arrow
call ispal("madam", 1, 3): return to line 9, purple arrow
call ispal("madam", 0, 4): return to main, line 17

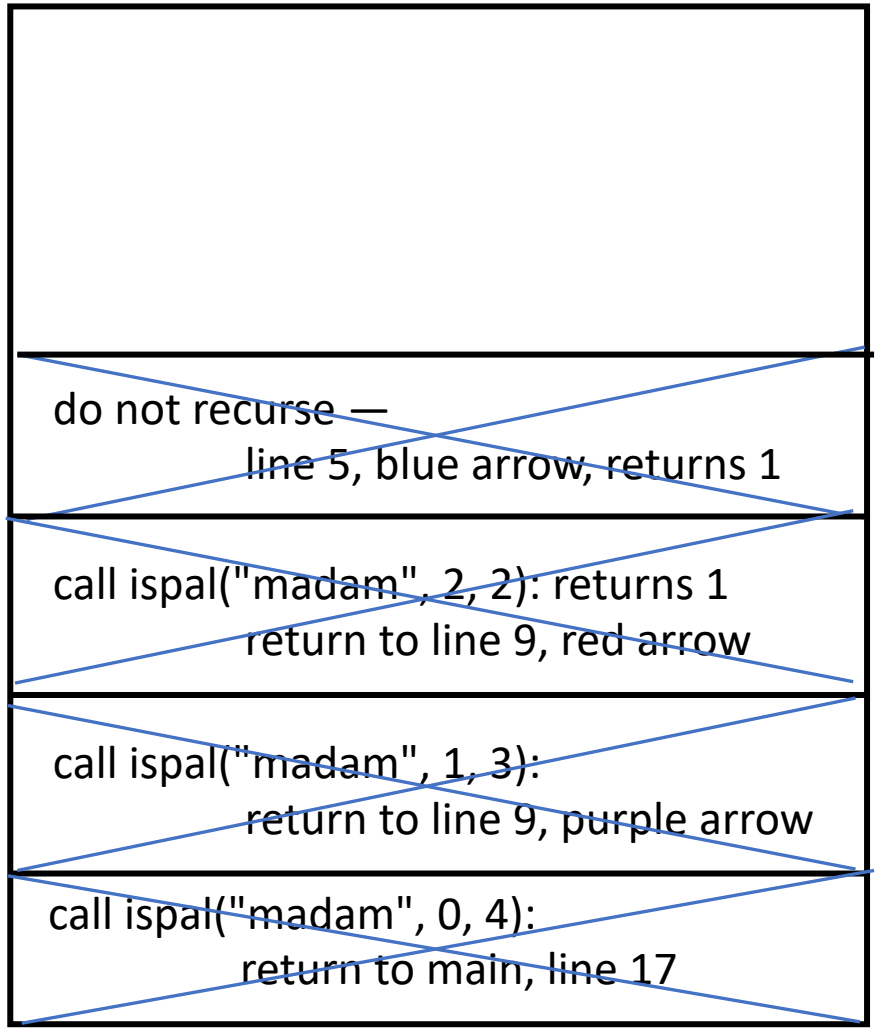
ispal(n ← "madam", b ← 1, e ← 3):
at line 9: return 1

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:            return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```



ispal(n ← "madam", b ← 0, e ← 4):
at line 9: return 1

```
 1:int ispal(char *n, int b, int e)
 2:{
 3:    /* base case */
 4:    if (b >= e)
 5:            return(1);
 6:
 7:    /* recursive case */
 8:    if (n[b] == n[e])
 9:        return(ispal(n, b+1, e-1));
10:    else return(0);
11:}
```



Greatest Common Divisor

- Find the largest integer that divides two other integers
  - Example: gcd(8, 12) = 4 as 8/4 = 2 and 12/4 = 3, and no larger number does that
  - Example: gcd(126, 28) = 14
- Euclid's Algorithm
  - gcd(1071, 462):
  - 1071 = 2 x 462 + 147
  - 462 = 3 x 147 + 21
  - 147 = 7 x 21 + 0
  - So gcd(1071, 462) = 21

```
 1:int gcd(int m, int n)
 2:{
 3:    int x;
 4:
 5:    /* base case: check for 0 */
 6:    if (n == 0) return(m);
 7:
 8:    /* recurse */
 9:    x = gcd(n, m % n);
10:
11:    /* done! */
12:    return(x);
13:}
```

```
14:
15:int main(void)
16:{
17:    int n;
18:
19:    n = gcd(4,6);
20:    printf("GCD of 4 and 6 is %d\n", n);
21:    return(0);
22:}
```

Initial call to gcd: gcd(m ← 4, n ← 6)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(4, 6): return to main, line 19
m = 4, n = 6

gcd(m ← 4, n ← 6) :

6: condition false, so skip

9: call gcd(6, 4)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(6, 4): return to line 9, purple arrow
m = 6, n = 4

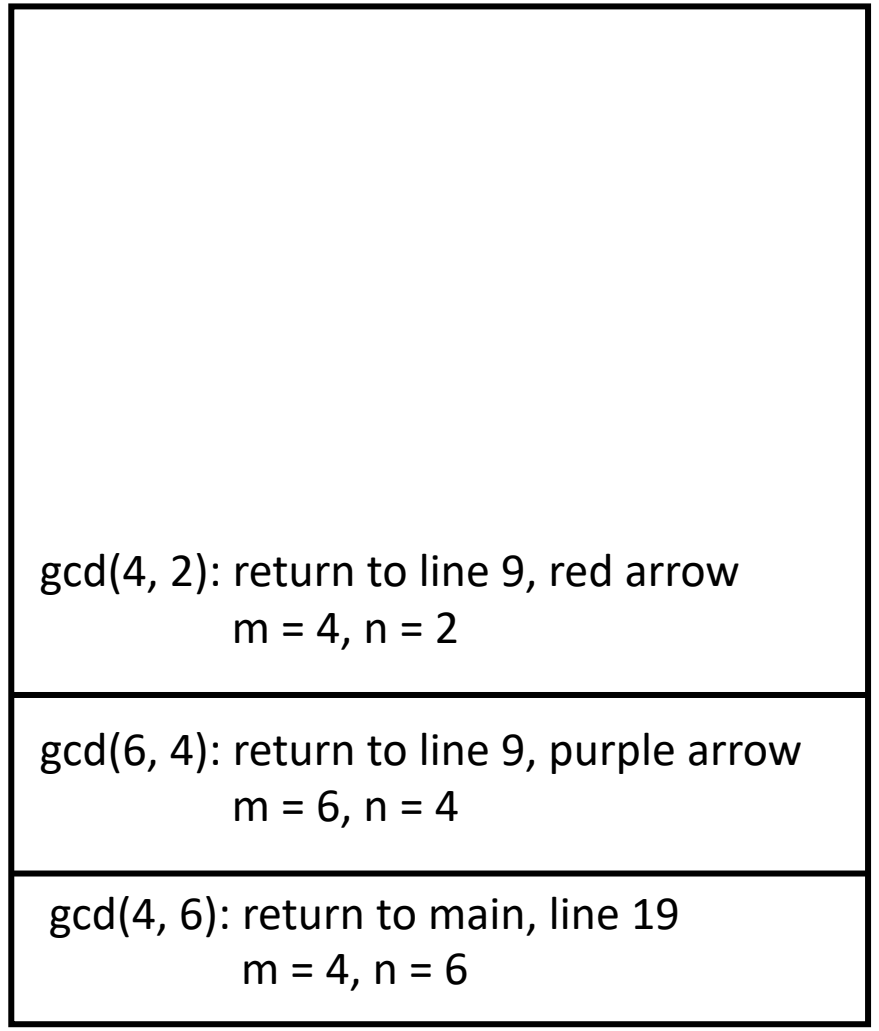
gcd(4, 6): return to main, line 19
m = 4, n = 6

gcd(m ← 6, n ← 4) :

6: condition false, so skip

9: call gcd(4, 2)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x =    gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```



gcd(m ← 4, n ← 2) :

6: condition false, so skip

9: call gcd(2, 0)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(2, 0): return to line 9, green arrow m = 2, n = 0
gcd(4, 2): return to line 9, red arrow m = 4, n = 2
gcd(6, 4): return to line 9, purple arrow m = 6, n = 4
gcd(4, 6): return to main, line 19 m = 4, n = 6

gcd(m ← 2, n ← 0) :

6: condition true, so return 2

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(1);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(2, 0): return to line 9, green arrow m = 2, n = 0; return 2
gcd(4, 2): return to line 9, red arrow m = 4, n = 2
gcd(6, 4): return to line 9, purple arrow m = 6, n = 4
gcd(4, 6): return to main, line 19 m = 4, n = 6

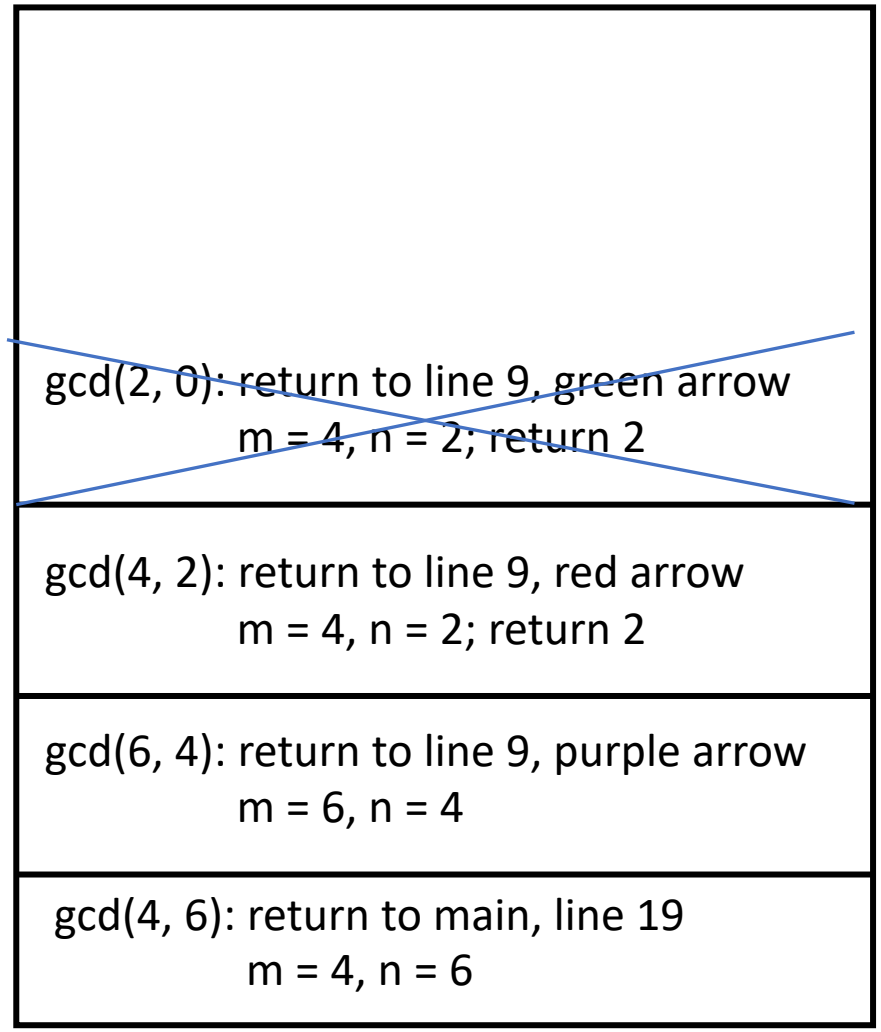
gcd(m ← 4, n ← 2) :

6: condition false, so skip

9: call gcd(2, 0); x = 2

12: return 2

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```



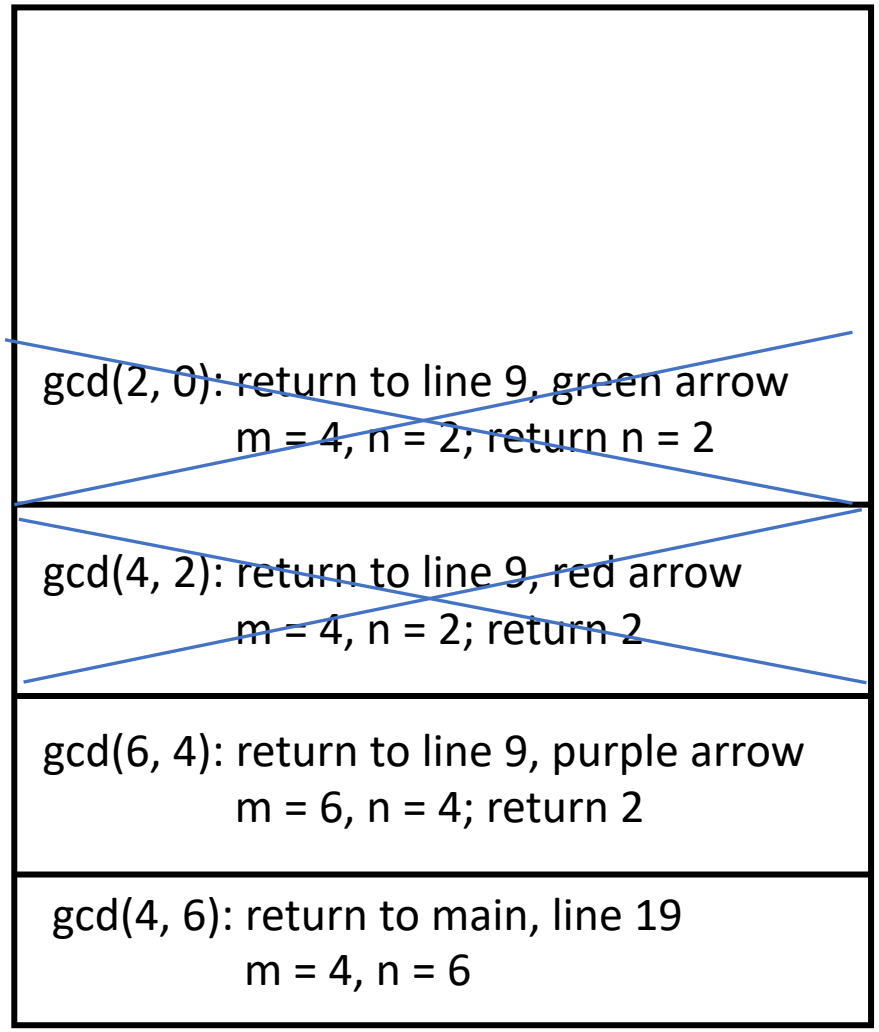
gcd(m ← 6, n ← 4) :

6: condition false, so skip

9: call gcd(4, 2); x = 2

12: return 2

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x =    gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```



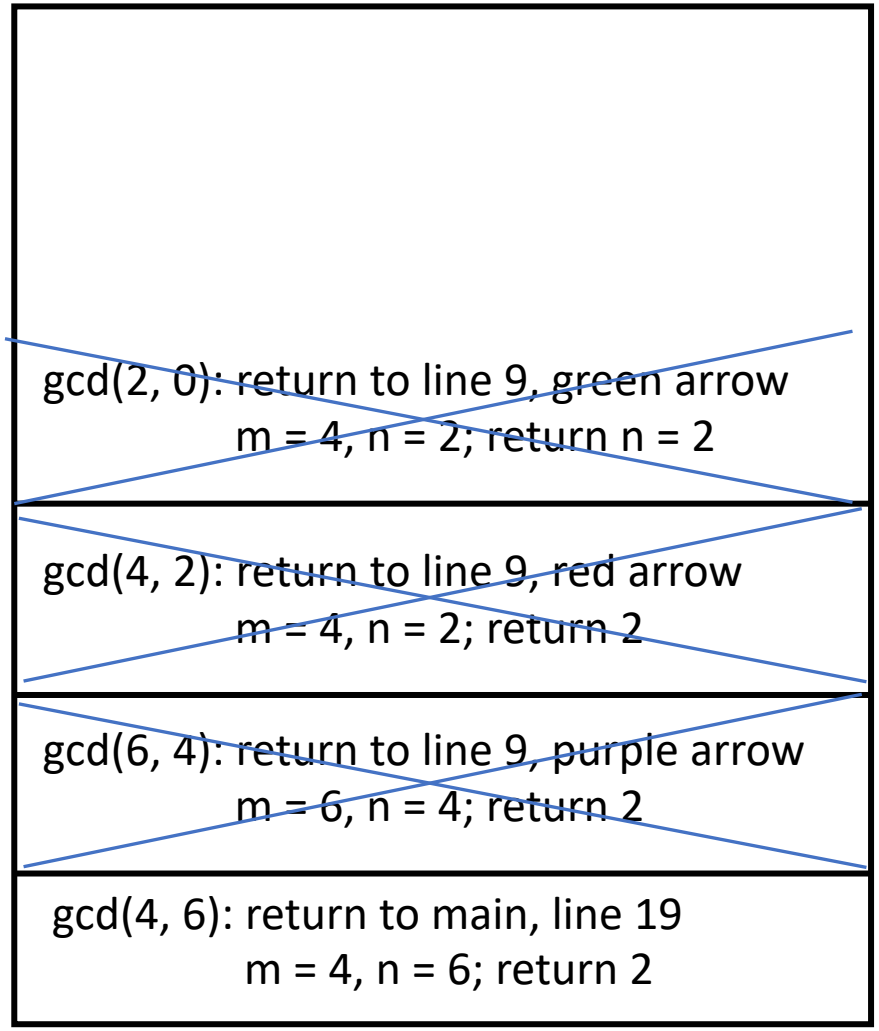
gcd(m ← 4, n ← 6) :

6: condition false, so skip

9: call gcd(6, 4); x = 2

12: return 2

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x =    gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```



```
 1:int gcd(int m, int n)
 2:{
 3:    int x;
 4:
 5:    /* base case: check for 0 */
 6:    if (n == 0) return(m);
 7:
 8:    /* recurse */
 9:    x = gcd(n, m % n);
10:
11:    /* done! */
12:    return(x);
13:}
```

```
14:
15:int main(void)
16:{
17:    int n;
18:
19:    n = gcd(126,28);
20:    printf("GCD of 126 and 28 is %d\n",
n);
21:    return(0);
22:}
```

Initial call to gcd: gcd(m ← 126, n ← 28)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(126, 28): return to main, line 19
m = 126, n = 28

gcd(m ← 126, n ← 28) :

6: condition false, so skip

9: call gcd(28, 14)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(28, 14): return to line 9, purple arrow
m = 28, n = 14

gcd(126, 28): return to main, line 19
m = 126, n = 28

gcd(m ← 28, n ← 14) :

6: condition false, so skip

9: call gcd(14, 0)

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(14, 0): return to line 9, red arrow m = 14, n = 0
gcd(28, 14): return to line 9, purple arrow m = 28, n = 14
gcd(126, 28): return to main, line 19 m = 126, n = 28

gcd(m ← 14, n ← 0) :

6: condition true, so return 14

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x = gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

gcd(14, 0): return to line 9, red arrow
m = 14, n = 0; return 14

gcd(28, 14): return to line 9, purple arrow
m = 28, n = 14

gcd(126, 28): return to main, line 19
m = 126, n = 28

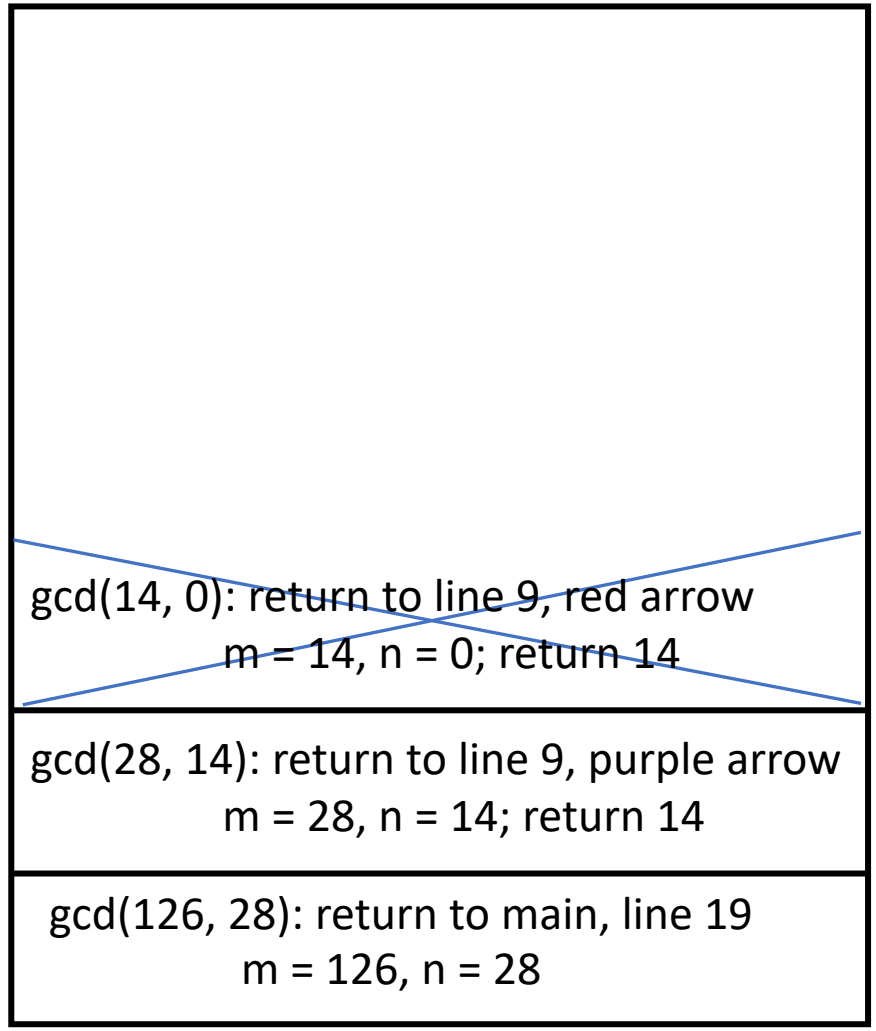
gcd(m ← 28, n ← 14) :

6: condition false, so skip

9: call gcd(14, 0); return 14

12: return 14

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x =    gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```



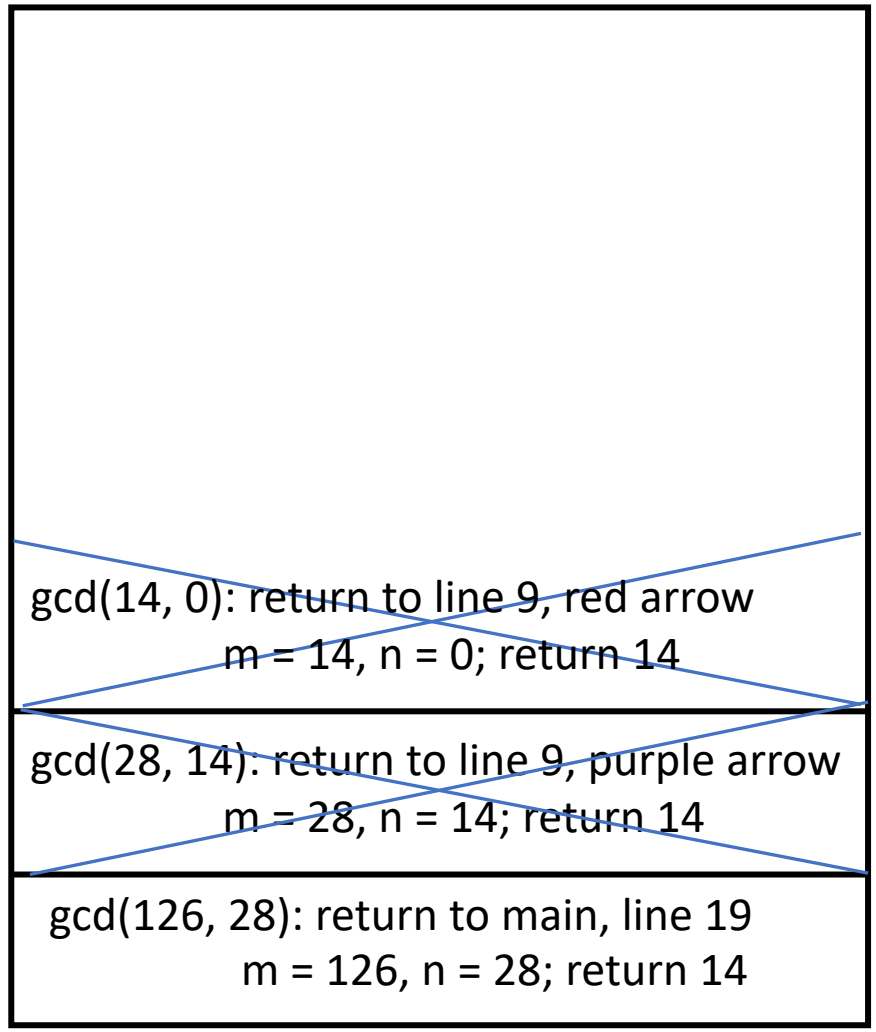
gcd(m ← 126, n ← 28) :

6: condition false, so skip

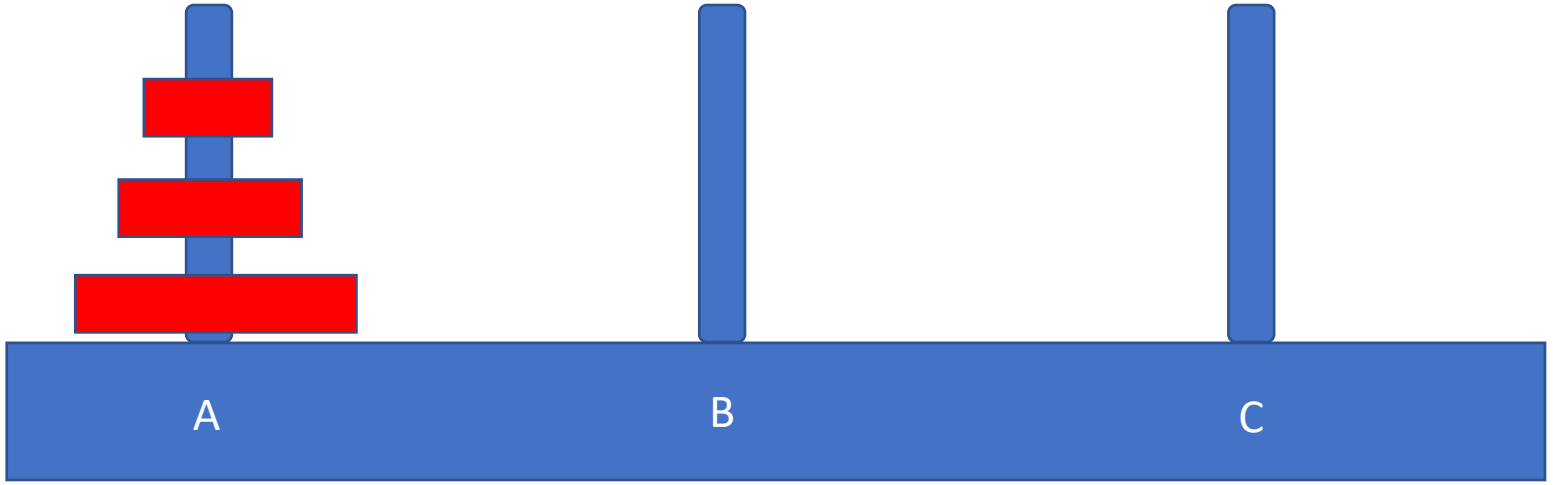
9: call gcd(28, 14); return 14

12: return 14

```
 1: int gcd(int m, int n)
 2: {
 3:     int x;
 4:
 5:     /* base case: check for 0 */
 6:     if (n == 0) return(m);
 7:
 8:     /* recurse */
 9:     x =    gcd(n, m % n);
10:
11:     /* done! */
12:     return(x);
13: }
```

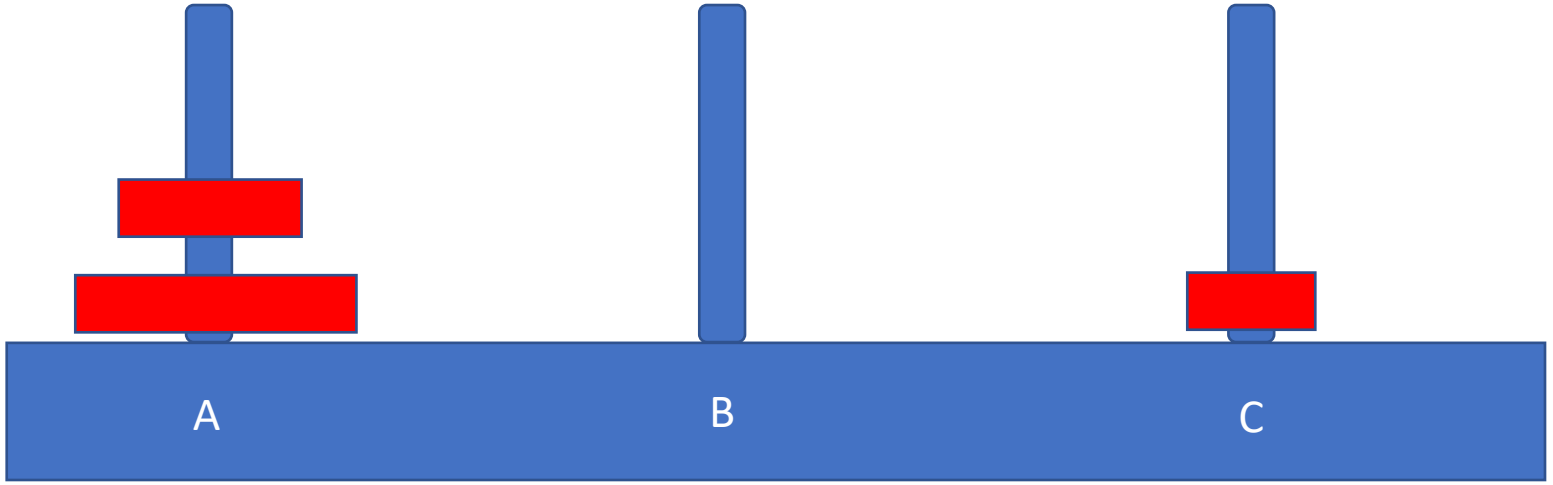


Tower of Hanoi



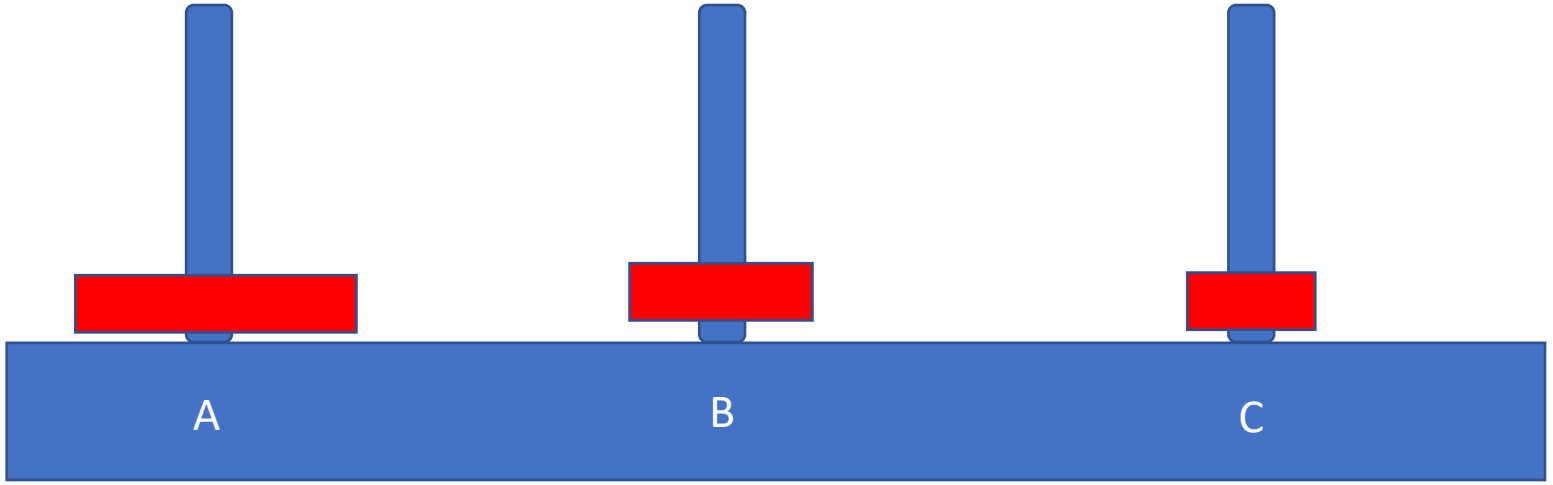
- Problem: move all 3 disks from peg A to peg C
- *Restriction:* can never put a larger disk on a smaller one!
- Approach: move all but bottom to peg B from peg A, move bottom one from A to C, then move stack from B to C

Tower of Hanoi



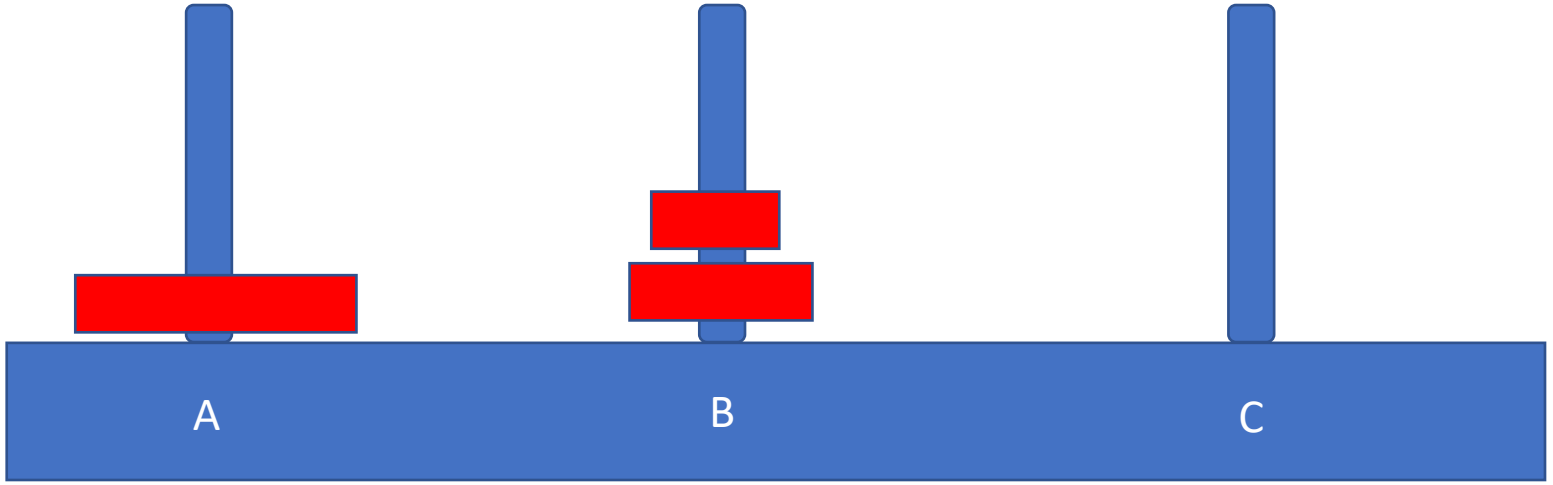
- Move top disk from A to C

Tower of Hanoi



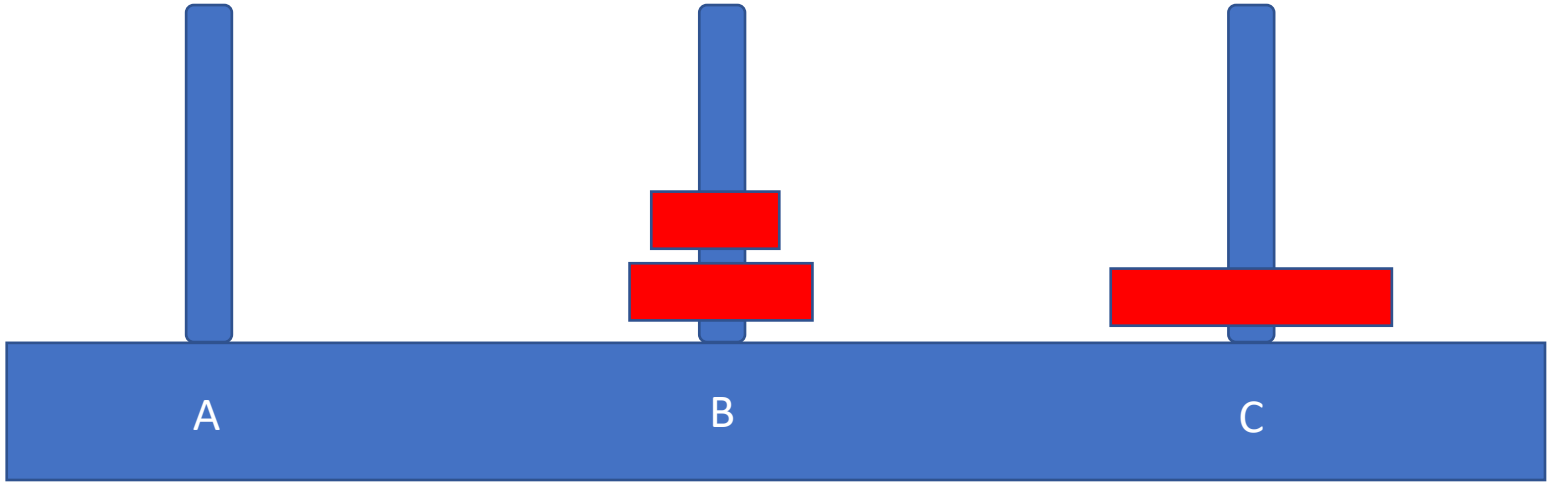
- Move top disk from A to B
- Now we can put C onto B and we have transferred the stack

Tower of Hanoi



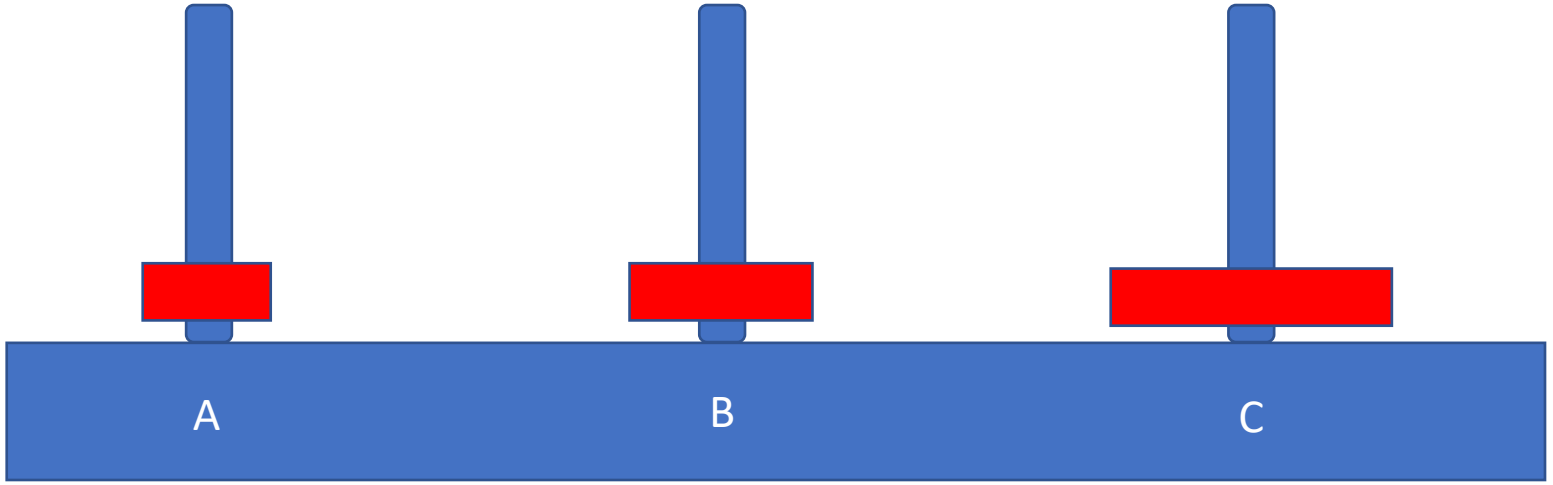
- Move top disk from C to B
- Stack is moved

Tower of Hanoi



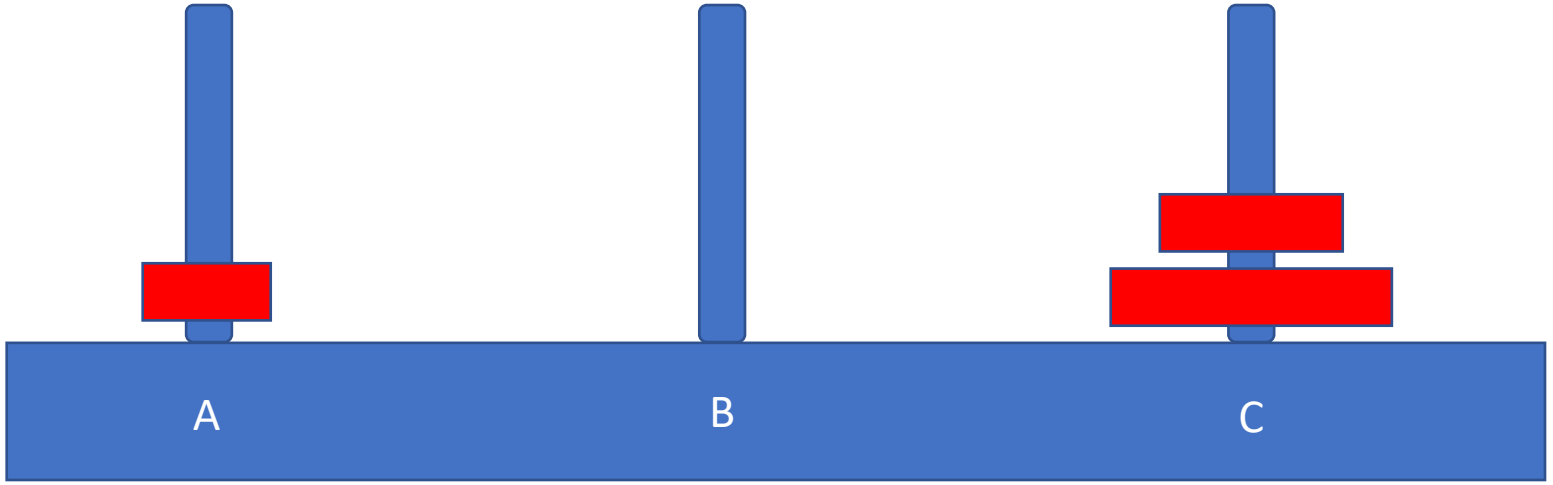
- Move top disk from A to B
- Now move disks from peg B to C
- Cannot do it directly; if we put the top one on peg C, we must move last peg from B to A. S put the top one on A

Tower of Hanoi



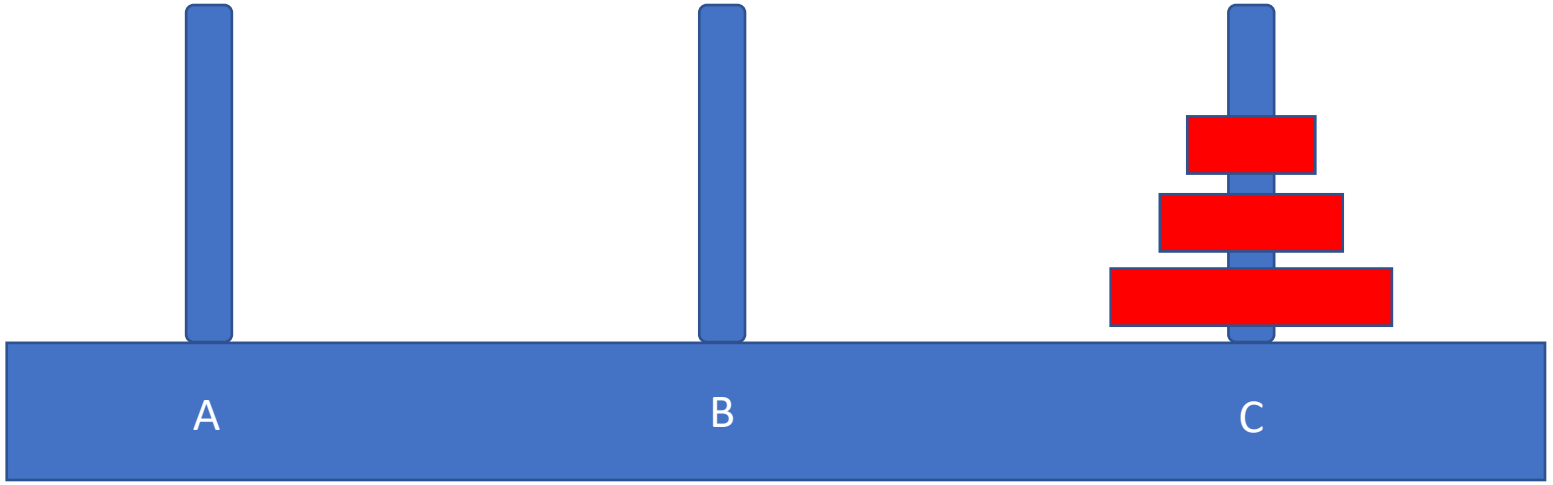
- Move top disk from A to B
- Now move disks from peg B to C
- Cannot do it directly; if we put the top one on peg C, we must move last peg from B to A. S put the top one on A

Tower of Hanoi



- Move top disk from B to C
- Almost done!

Tower of Hanoi



- Move top disk from A to C
- And done!

Sequence of Moves

- Move top disk from tower A to tower C
- Move top disk from tower A to tower B
- Move top disk from tower C to tower B
- Move top disk from tower A to tower C
- Move top disk from tower B to tower A
- Move top disk from tower B to tower C
- Move top disk from tower A to tower C

Sequence of Moves

- Move top disk from tower A to tower C
- Move top disk from tower A to tower B
- Move top disk from tower C to tower B

Move top 2 disks from A to B

- Move top disk from tower A to tower C

- Move top disk from tower B to tower A
- Move top disk from tower B to tower C
- Move top disk from tower A to tower C

Move top 2 disks from B to C

```
int tower(int ndisks, char fromtower, char totower, char temptower)
{
        int n; /* number of disks moved */

        /* base case: move 1 disk */
        if (ndisks == 1){
                printf("Move top disk from tower %c to tower %c\n",
                                                        fromtower, totower);
                return(1);
        }
        /* now recurse */
        n = tower(ndisks-1, fromtower, temptower, totower);
        n += tower(1, fromtower, totower, temptower);
        n += tower(ndisks-1,  temptower, totower, fromtower);

        /* return the number of disks moved */
        return(n);
}
```