ECS 36A, April 26, 2023

Announcements

- TA’s new office hours on Friday are 12:00 noon – 1:00pm
- Tutoring is available from the CS Tutoring Club

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:}
```

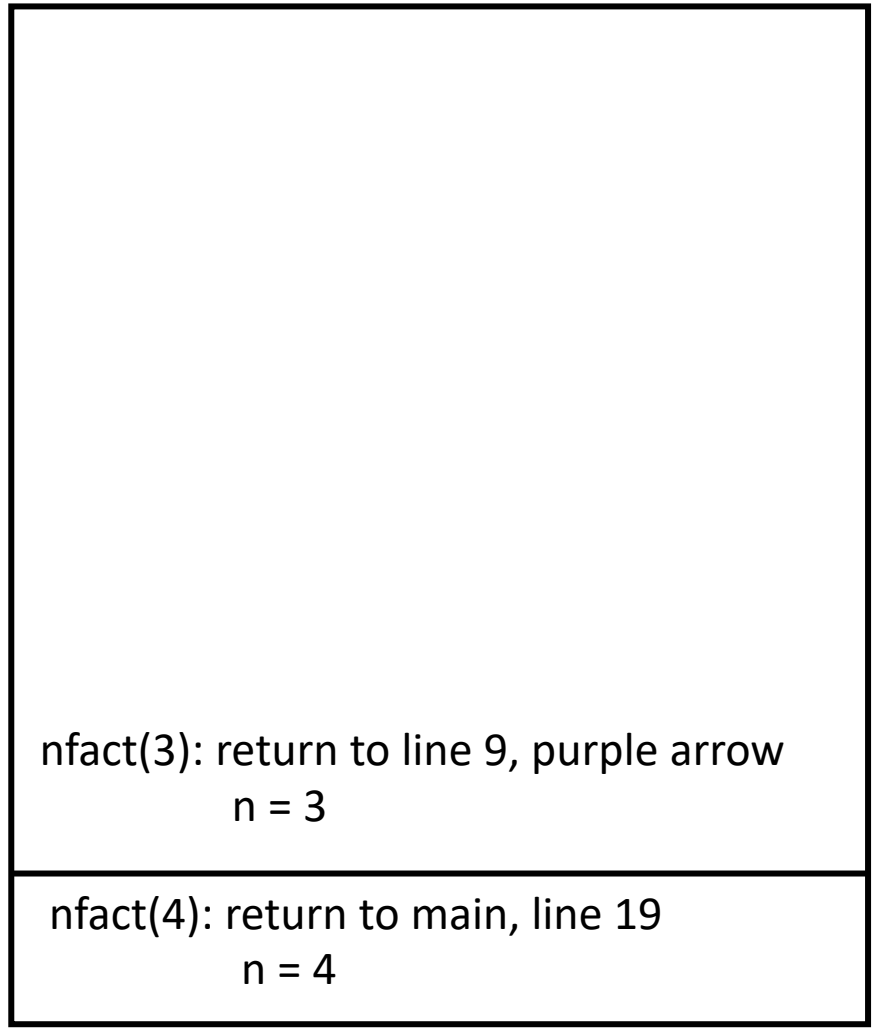
nfact(4): return to main, line 19
n = 4

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: }
```

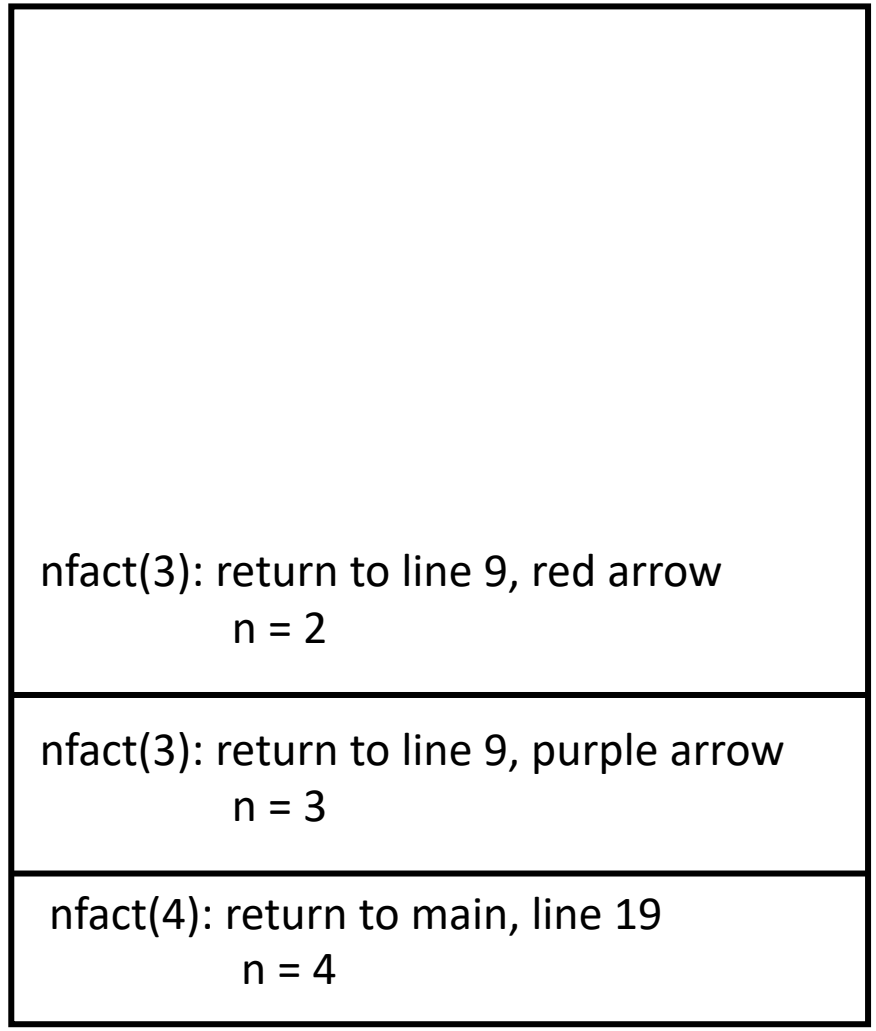


nfact(n ← 3):

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

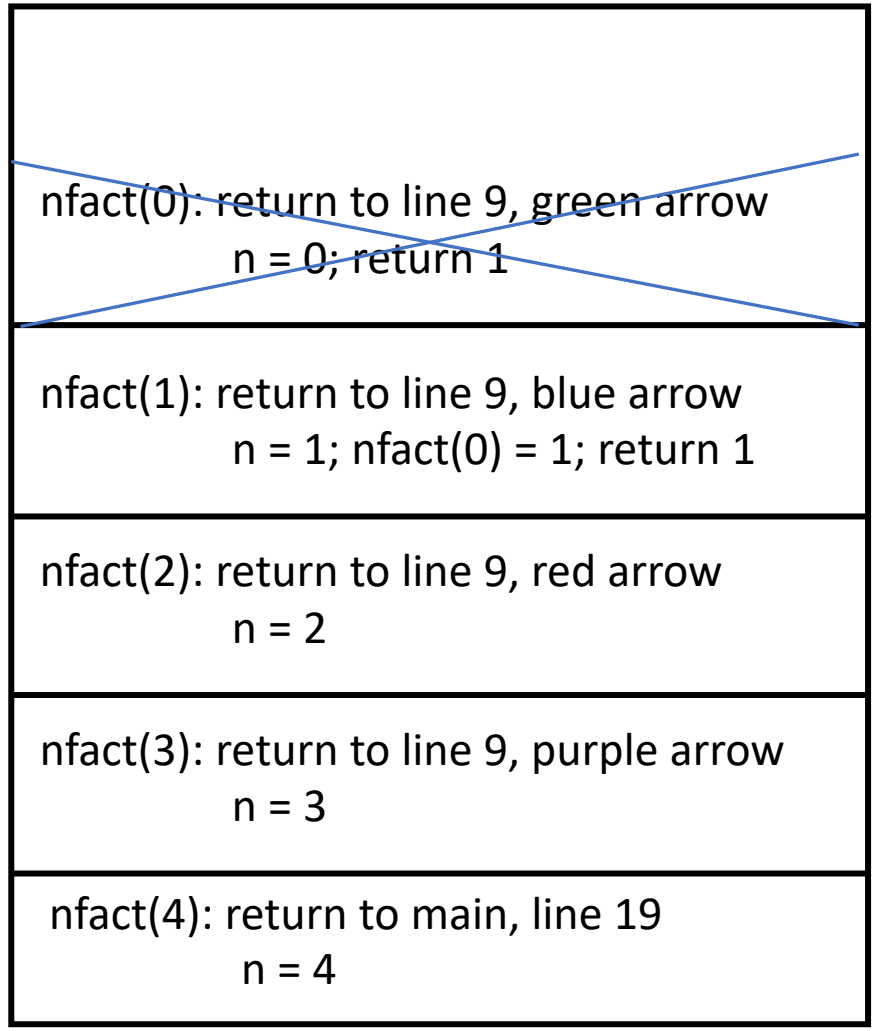
nfact(n ← 1):

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: }
```



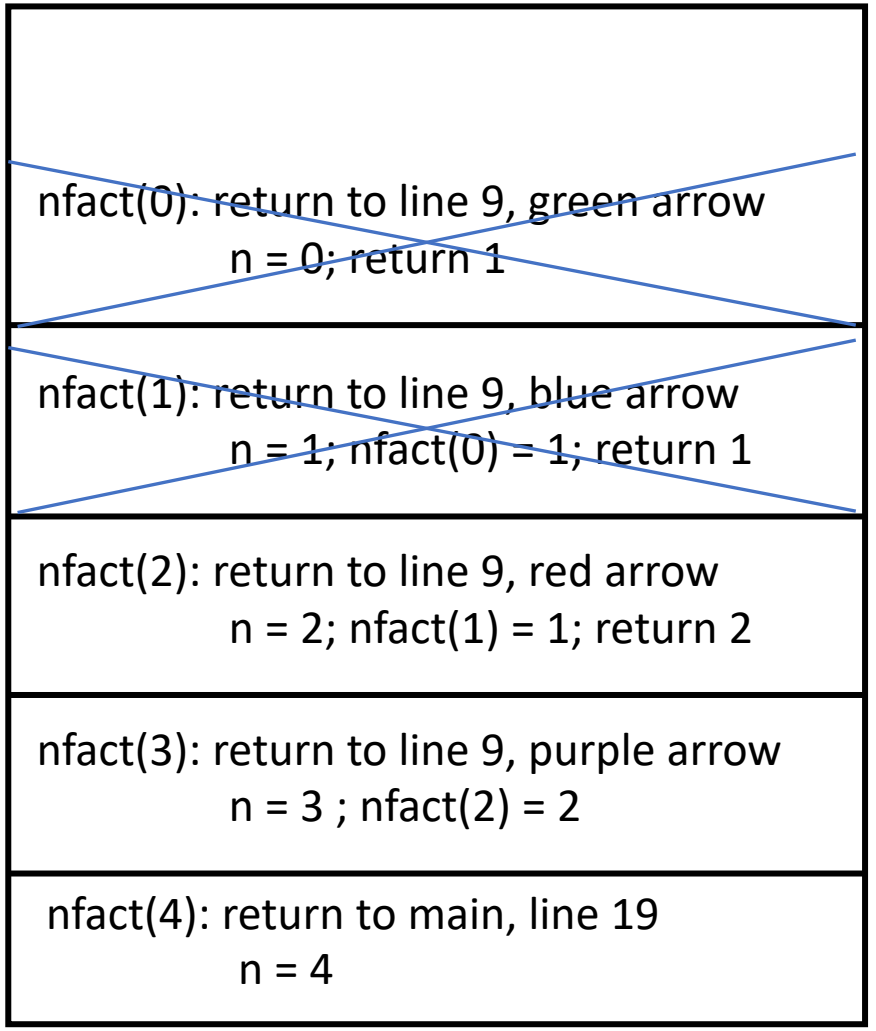
nfact(n ← 2):

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: }
```



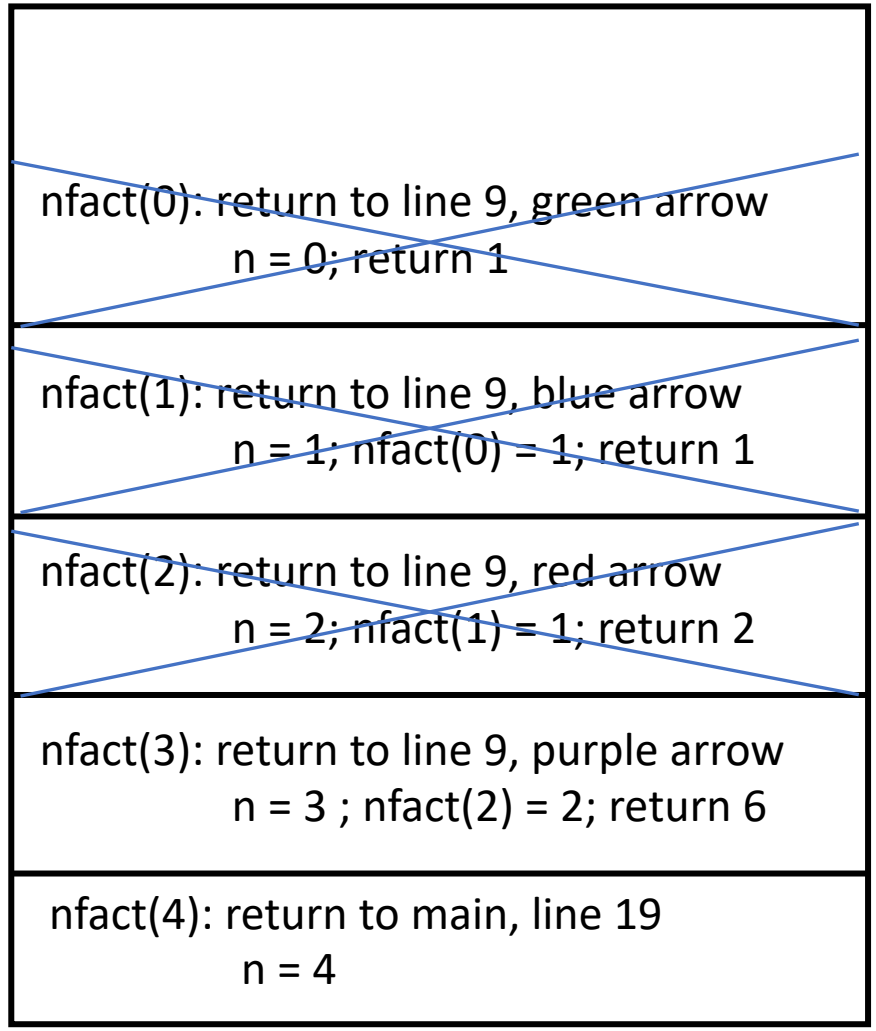
nfact(n ← 3):

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: }
```



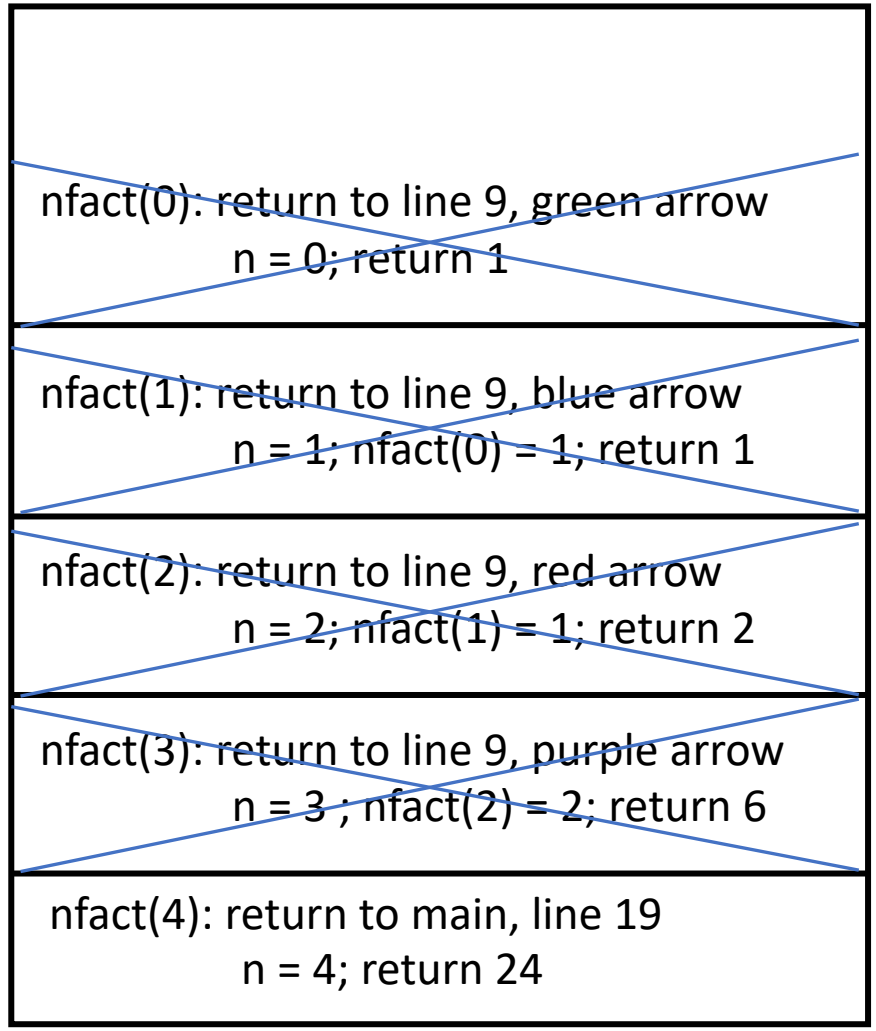
nfact(n ← 4):

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”

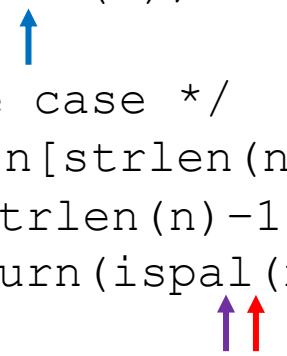
ispal(“ada”): return to line 10, purple arrow
n = “ada”

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”

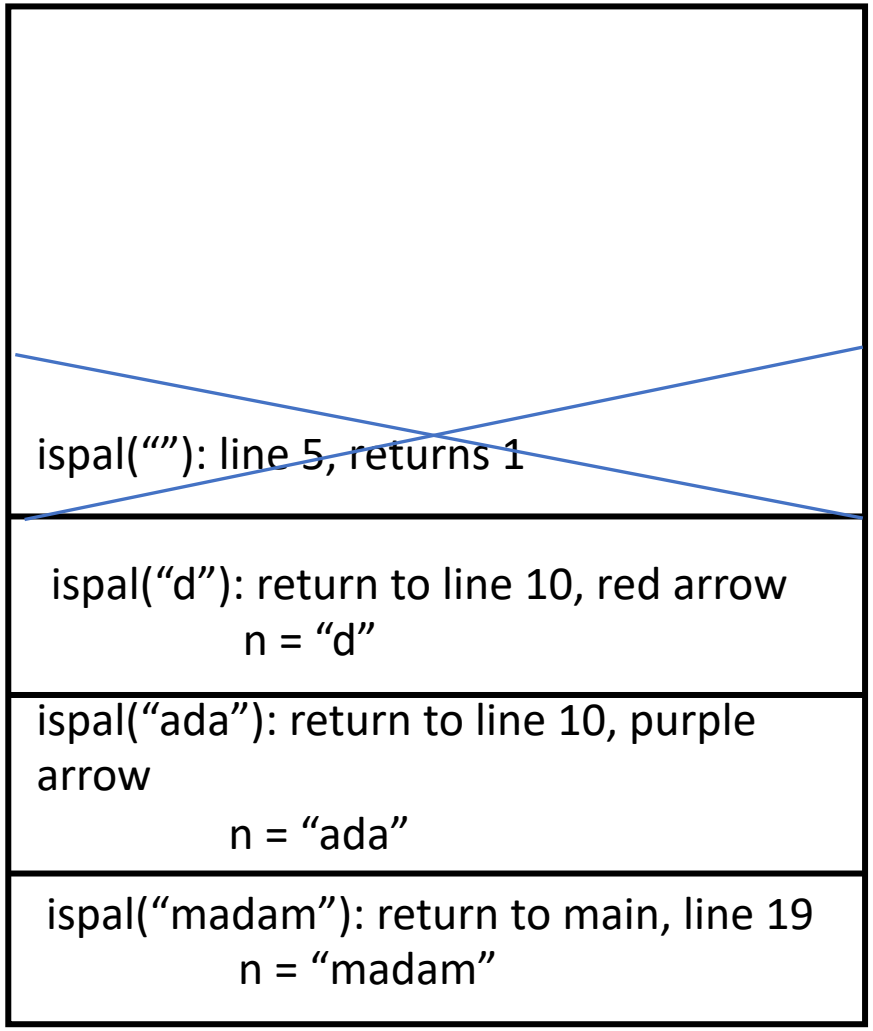
ispal(“madam”): return to main, line 19

n = “madam”

ispal(n ← “d”):

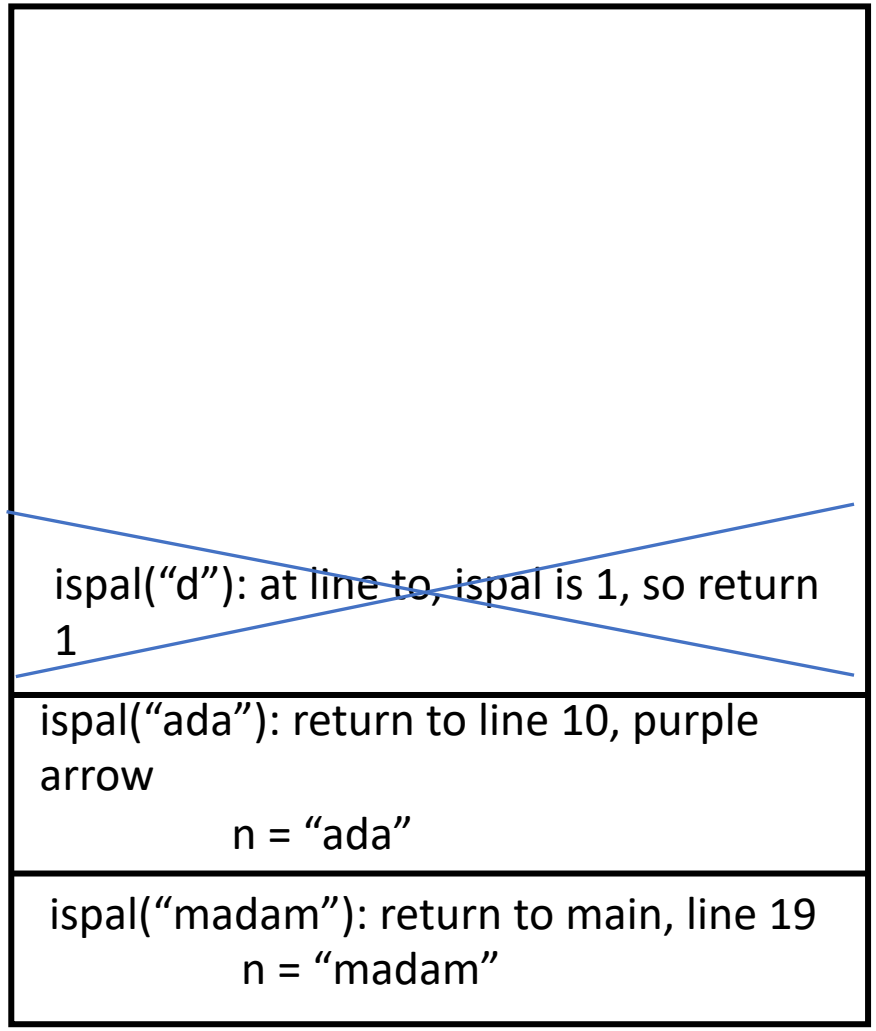
6: condition true, so return 1

```
 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:}
```



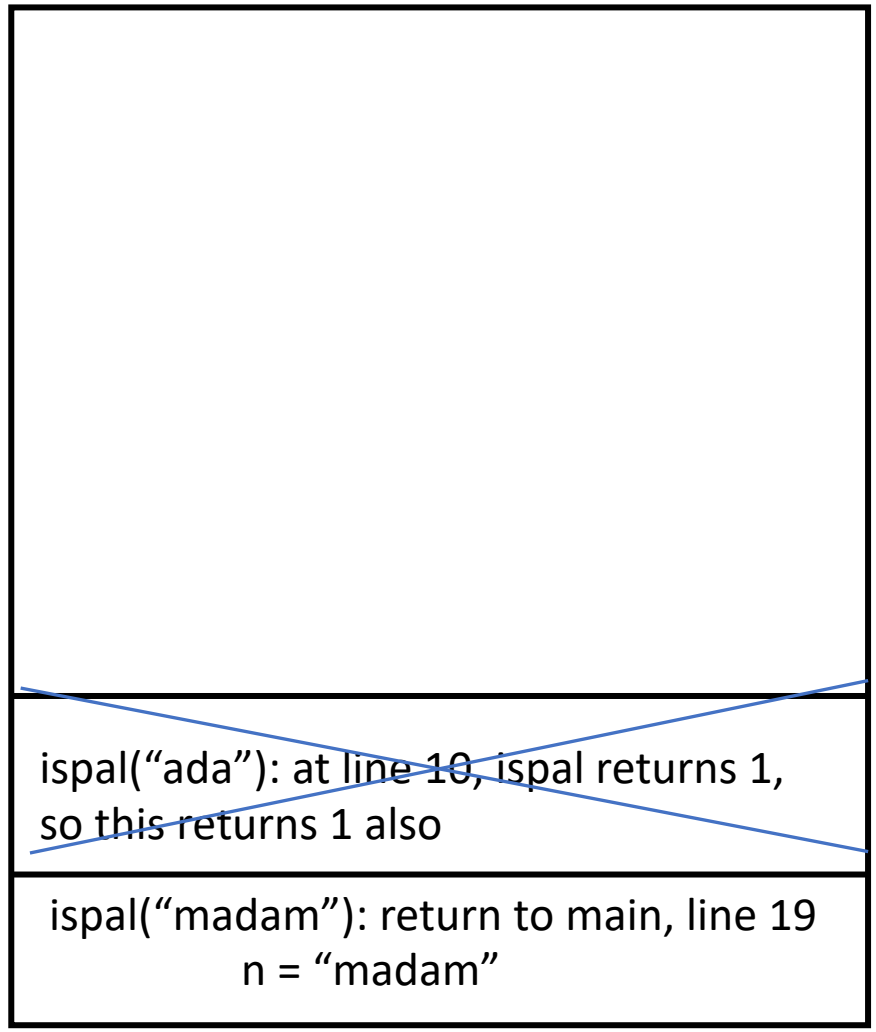
ispal(n ← “d”):
at line 10, return 1

```
 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:}
```