n! Done Recursively

Matt Bishop

ECS 36A, Fall Quarter 2019

UC Davis

```
 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:}
```

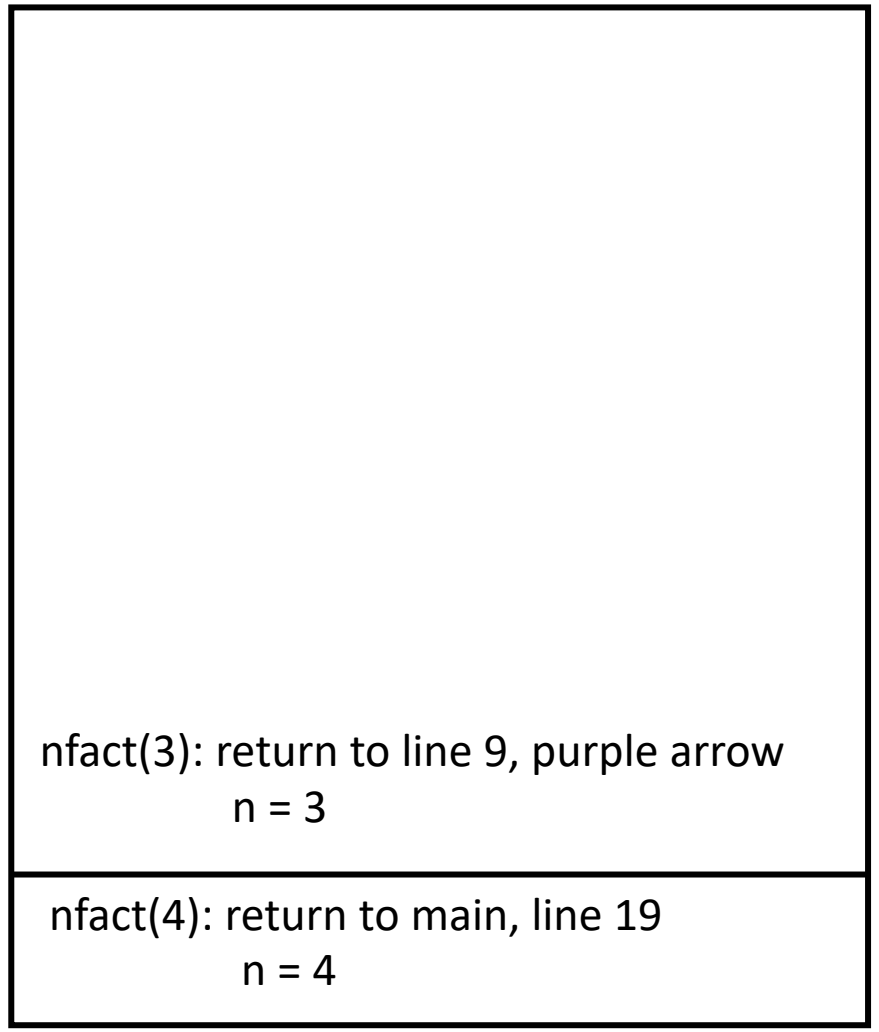
nfact(4): return to main, line 19
n = 4

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: }
```

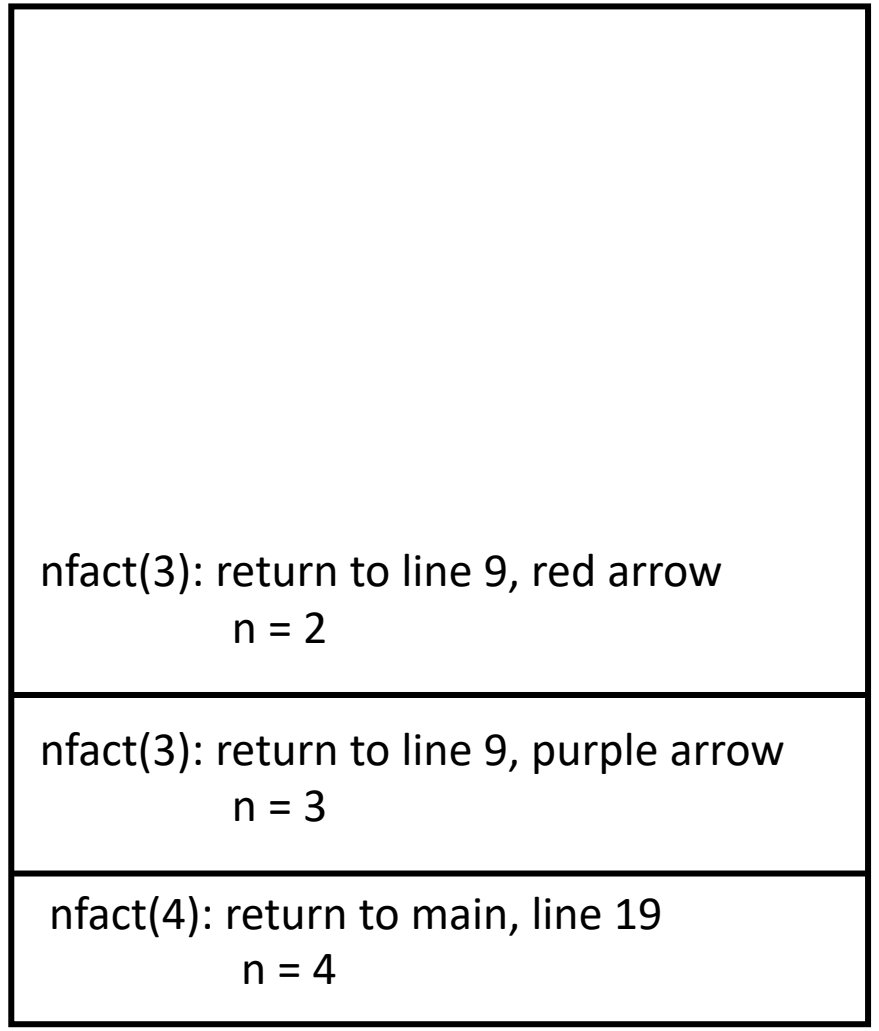


nfact(n ← 3):

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

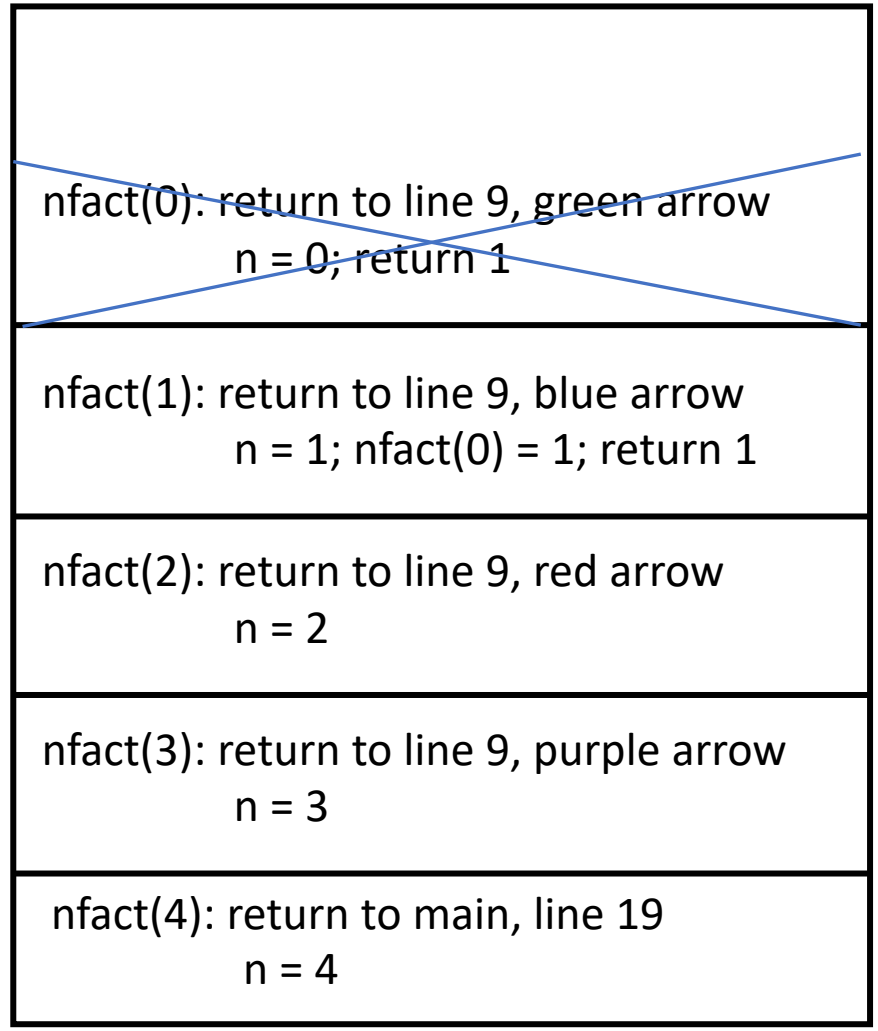
nfact(n ← 1):

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: }
```



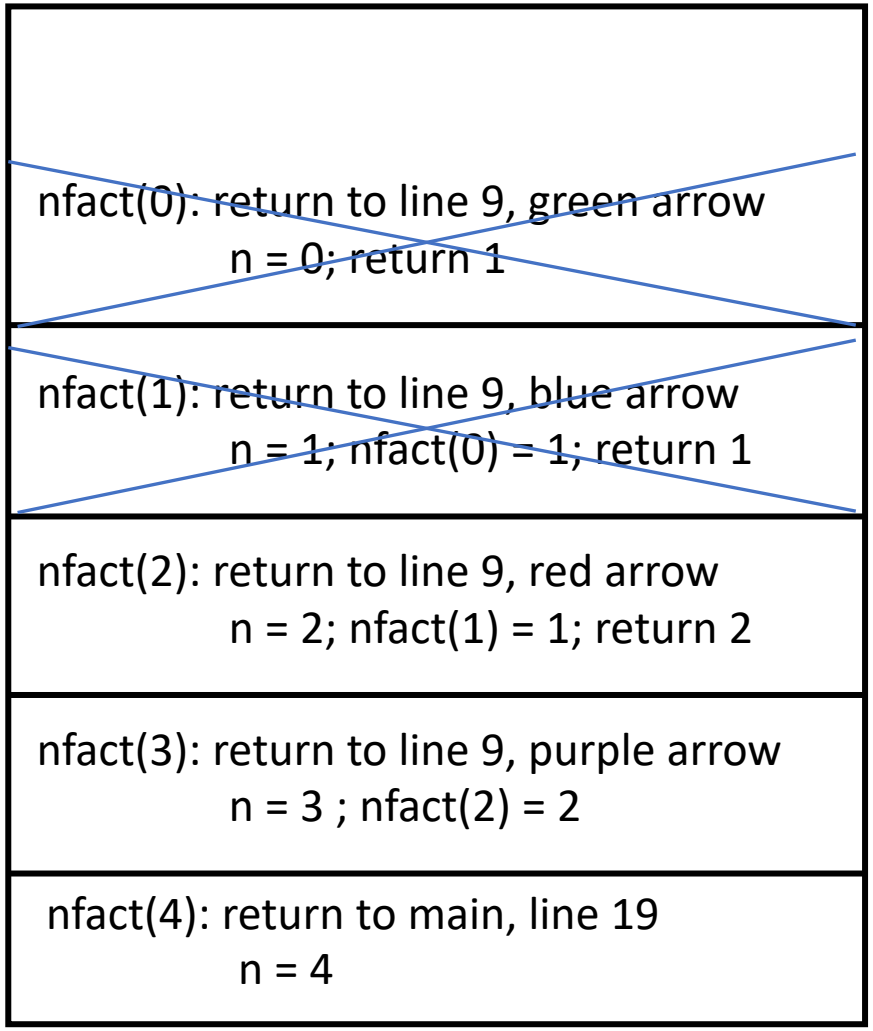
nfact(n ← 2):

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: }
```



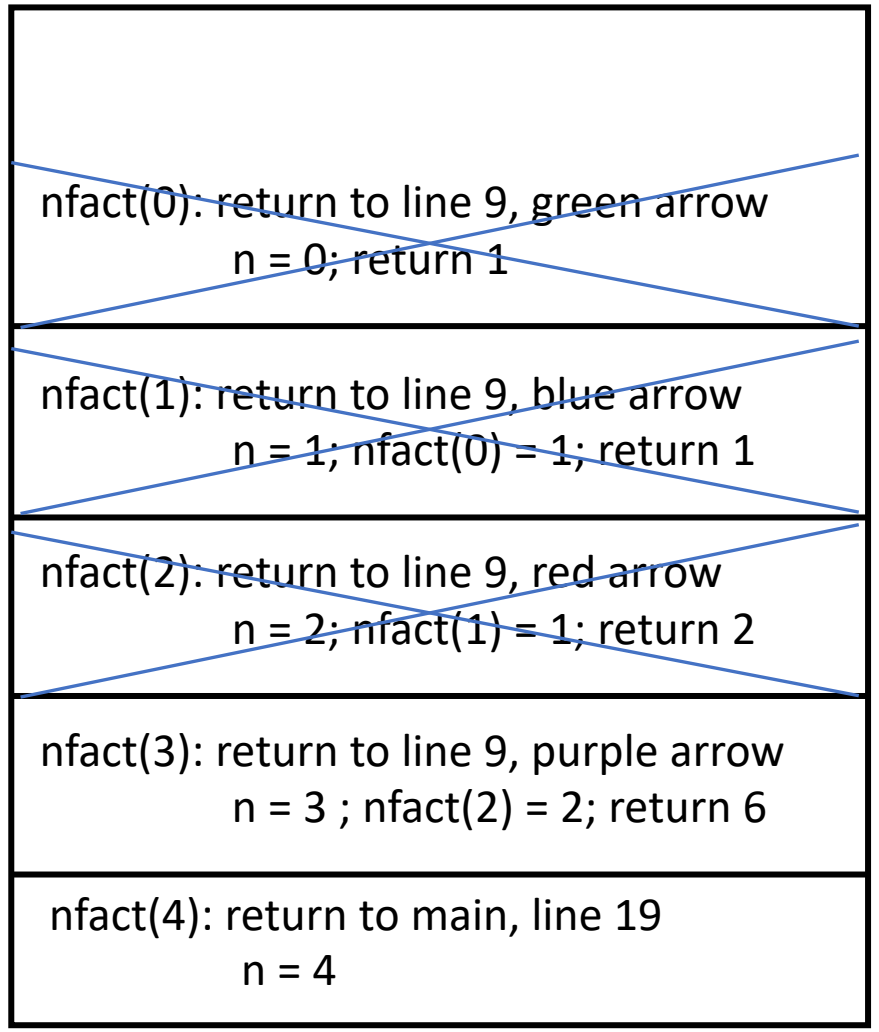
nfact(n ← 3):

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: }
```