Core Design Concepts Discussed:

Analysis

Evaluation

Performance and Optimization of Recursive Functions

Harley Eades III

The Performance Hit

Consider evaluating the following recursive function:

```
let rec mult m n =
 if m == 0
 then 0
 else if n == 0
      then 0
      else let rc = mult m (n - 1) in
           let ret = m + rc in
           ret

 let main =
   let m = 1 in
    let n = 2 in
      let answ = mult m n in
          answ

 main;;
```

```
1. let rec mult m n =
2.  if m == 0
3.  then 0
4.  else if n == 0
5.       then 0
6.       else let rc = mult m (n - 1) in
7.            let ret = m + rc in
8.            ret
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>

```
1. let rec mult m n =
2.  if m == 0
3.  then 0
4.  else if n == 0
5.       then 0
6.       else let rc = mult m (n - 1) in
7.            let ret = m + rc in
8.            ret
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?

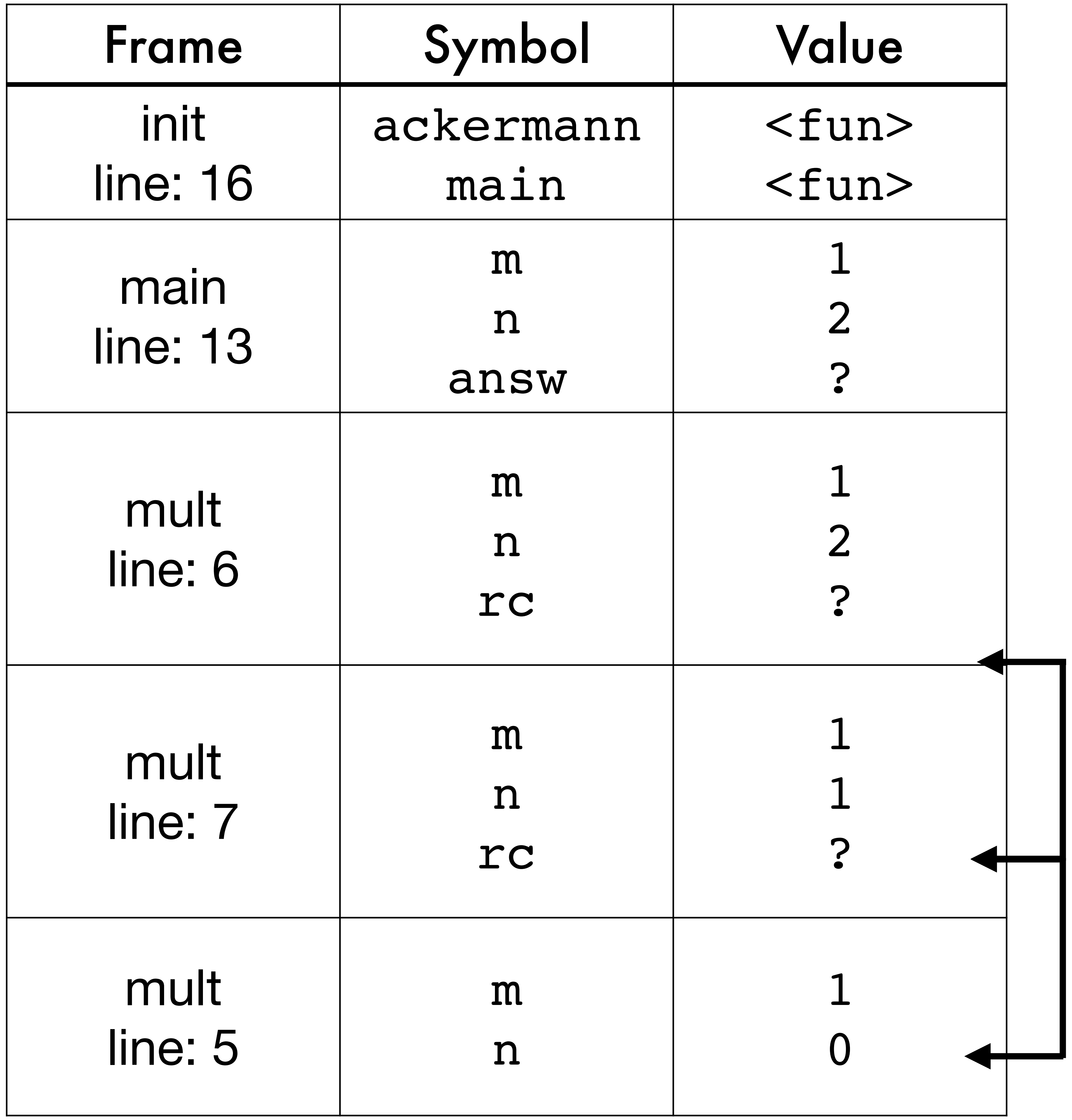
```
   1. let rec mult m n =
   2.  if m == 0
   3.  then 0
   4.  else if n == 0
   5.       then 0
→  6.       else let rc = mult m (n - 1) in
   7.            let ret = m + rc in
   8.            ret
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult: line 6	m n rc	1 2 ?

```
1. let rec mult m n =
2.  if m == 0
3.  then 0
4.  else if n == 0
5.       then 0
6.       else let rc = mult m (n - 1) in
7.            let ret = m + rc in
8.            ret
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult: line 6	m n rc	1 2 ?
mult: line 7	m n ret	1 1 ?

```
   1. let rec mult m n =
   2.  if m == 0
   3.  then 0
   4.  else if n == 0
   5.       then 0
→  6.       else let rc = mult m (n - 1) in
   7.            let ret = m + rc in
   8.            ret
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```



Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 6	m n rc	1 2 ?
mult line: 7	m n rc	1 1 ?
mult line: 5	m n	1 0

```
   1. let rec mult m n =
   2.  if m == 0
   3.  then 0
   4.  else if n == 0
→  5.      then 0
   6.      else let rc = mult m (n - 1) in
   7.           let ret = m + rc in
   8.           ret
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 6	m n rc	1 2 ?
mult line: 7	m n ret	1 1 0
mult line: 5	m n	1 0

```
1. let rec mult m n =
2.  if m == 0
3.  then 0
4.  else if n == 0
5.       then 0
6.       else let rc = mult m (n - 1) in
7.            let ret = m + rc in
8.            ret
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 6	m n rc	1 2 1
mult line: 7	m n rc	1 1 0
mult line: 5	m n	1 0

```
   1. let rec mult m n =
   2.  if m == 0
   3.  then 0
   4.  else if n == 0
→  5.       then 0
   6.       else let rc = mult m (n - 1) in
   7.            let ret = m + rc in
   8.            ret
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	ackermann main	<fun> <fun>
main line: 13	m n answ	1 2 2
mult line: 6	m n rc	1 2 1
mult line: 7	m n rc	1 1 0
mult line: 5	m n	1 0

```
1. let rec mult m n =
2.  if m == 0
3.  then 0
4.  else if n == 0
5.       then 0
6.       else let rc = mult m (n - 1) in
7.            let ret = m + rc in
8.            ret
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

- Bad for performance: making a recursive call in an argument position (line 7).

- This results in the bindings of an activation record depending on the return value of a new activation record.

- Thus, the compiler will create lots of activation records that cannot be popped off of the stack until the end of evaluation.

- This results in a bad use of memory.

Tail Recursion using the accumulator pattern

Non-tail recursive:

```
let rec mult m n =
 if m == 0
 then 0
 else if n == 0
      then 0
      else m + (mult m (n - 1))
```

Tail recursive:

```
let rec mult_helper acc m n =
  if m == 0
  then 0
  else if n == 0
       then acc
       else mult_helper (m + acc) m (n - 1)

let mult m n = mult_helper 0 m n
```

Evaluation of Tail Recursion

```
let mult m n =
  let rec mult_helper acc n' =
    if m == 0
    then 0
    else if n == 0
         then acc
         else mult_helper (m + acc) (n - 1)
  in mult_helper 0 n

 let main =
   let m = 1 in
    let n = 2 in
      let answ = mult m n in
          answ

 main;;
```

```
1. let mult m n =
2.   let rec mult_helper acc n' =
3.     if m == 0
4.     then 0
5.     else if n == 0
6.          then acc
7.          else mult_helper (m + acc) (n - 1)
8.   in mult_helper 0 n
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init	mult	<fun>
line: 16	main	<fun>

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
   7.          else mult_helper (m + acc) (n - 1)
   8.   in mult_helper 0 n
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
→  13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?

```
  1. let mult m n =
  2.   let rec mult_helper acc n' =
  3.     if m == 0
  4.     then 0
  5.     else if n == 0
  6.          then acc
  7.          else mult_helper (m + acc) (n - 1)
→ 8.   in mult_helper 0 n
  9.
  10. let main =
  11.   let m = 1 in
  12.    let n = 2 in
  13.      let answ = mult m n in
  14.          answ
  15.
  16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 8	m n	1 2

```
  1. let mult m n =
  2.   let rec mult_helper acc n' =
  3.     if m == 0
  4.     then 0
  5.     else if n == 0
  6.          then acc
→ 7.          else mult_helper (m + acc) (n - 1)
  8.   in mult_helper 0 n
  9.
  10. let main =
  11.   let m = 1 in
  12.    let n = 2 in
  13.      let answ = mult m n in
  14.          answ
  15.
  16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 8	m n	1 2
mult_helper line: 7	m n acc n'	1 2 0 2

```
1. let mult m n =
2.   let rec mult_helper acc n' =
3.     if m == 0
4.     then 0
5.     else if n == 0
6.          then acc
7.          else mult_helper (m + acc) (n - 1)
8.   in mult_helper 0 n
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 8	m n	1 2
mult_helper line: 7	m n acc n'	1 2 0 2
mult_helper line: 7	m n acc n'	1 2 1 1

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
→  7.          else mult_helper (m + acc) (n - 1)
   8.   in mult_helper 0 n
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.           answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 8	m n	1 2
mult_helper line: 7	m n acc n'	1 2 0 2
mult_helper line: 7	m n acc n'	1 2 1 1
mult_helper line: 7	m n acc n'	1 2 2 0

```
  1. let mult m n =
  2.   let rec mult_helper acc n' =
  3.     if m == 0
  4.     then 0
  5.     else if n == 0
  6.          then acc
→ 7.          else mult_helper (m + acc) (n - 1)
  8.   in mult_helper 0 n
  9.
  10. let main =
  11.   let m = 1 in
  12.    let n = 2 in
  13.      let answ = mult m n in
  14.          answ
  15.
  16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 2
mult line: 8	m n	1 2
mult_helper line: 7	m n acc n'	1 2 0 2
mult_helper line: 7	m n acc n'	1 2 1 1
mult_helper line: 7	m n acc n'	1 2 2 0

Optimization: Tail Recursion

```
let mult m n =
  let rec mult_helper acc n' =
    if m == 0
    then 0
    else if n == 0
         then acc
         else mult_helper (m + acc) (n - 1)
  in mult_helper 0 n

let main =
  let m = 1 in
   let n = 2 in
     let answ = mult m n in
         answ

main;;
```

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
   7.          else mult_helper (m + acc) (n - 1)
   8.   in mult_helper 0 n
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
→  16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>

```
1. let mult m n =
2.   let rec mult_helper acc n' =
3.     if m == 0
4.     then 0
5.     else if n == 0
6.          then acc
7.          else mult_helper (m + acc) (n - 1)
8.   in mult_helper 0 n
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
   7.          else mult_helper (m + acc) (n - 1)
→  8.   in mult_helper 0 n
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult line: 8	m n	1 2

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
→  7.          else mult_helper (m + acc) (n - 1)
   8.   in mult_helper 0 n
   9.
   10. let main =
   11.   let m = 1 in
   12.    let n = 2 in
   13.      let answ = mult m n in
   14.          answ
   15.
   16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult_helper line: 7	m n acc n'	1 2 0 2

```
1. let mult m n =
2.   let rec mult_helper acc n' =
3.     if m == 0
4.     then 0
5.     else if n == 0
6.          then acc
7.          else mult_helper (m + acc) (n - 1)
8.   in mult_helper 0 n
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

→ 7.

Frame	Symbol	Value
init line: 16	mult	<fun>
	main	<fun>
main line: 13	m	1
	n	2
	answ	?
mult_helper line: 7	m	1
	n	2
	acc	1
	n'	1

```
1. let mult m n =
2.   let rec mult_helper acc n' =
3.     if m == 0
4.     then 0
5.     else if n == 0
6.          then acc
→ 7.          else mult_helper (m + acc) (n - 1)
8.   in mult_helper 0 n
9.
10. let main =
11.   let m = 1 in
12.    let n = 2 in
13.      let answ = mult m n in
14.          answ
15.
16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 ?
mult_helper line: 7	m n acc n'	1 2 2 0

```
   1. let mult m n =
   2.   let rec mult_helper acc n' =
   3.     if m == 0
   4.     then 0
   5.     else if n == 0
   6.          then acc
→  7.          else mult_helper (m + acc) (n - 1)
   8.   in mult_helper 0 n
   9.
  10. let main =
  11.   let m = 1 in
  12.    let n = 2 in
  13.      let answ = mult m n in
  14.          answ
  15.
  16. main;;
```

Frame	Symbol	Value
init line: 16	mult main	<fun> <fun>
main line: 13	m n answ	1 2 2

Tail Call Optimization

- Tail calls do not require any modifications to the activation frame. Thus, we do not need to keep them around.
- Compiler can detect tail recursion, and then optimize its stack usage by discarding each activation frame during evaluation.
  - Constant space usage!
  - The same performance as loops!
- Not all PLs offer this tail call optimization!

Tail Call Optimization

PL	Tail Call Optimized	Compiler
C/C++	Yes	GCC
Swift	Yes	All
Python	No	All
C#	No	All
Java	Partially	JVM
OCaml	Yes	All
Haskell	Yes	GHC
javascript	Yes	ES6