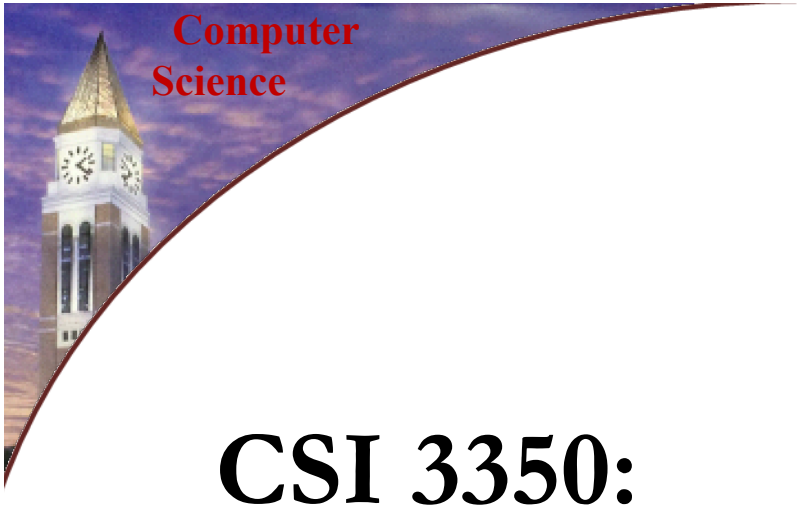
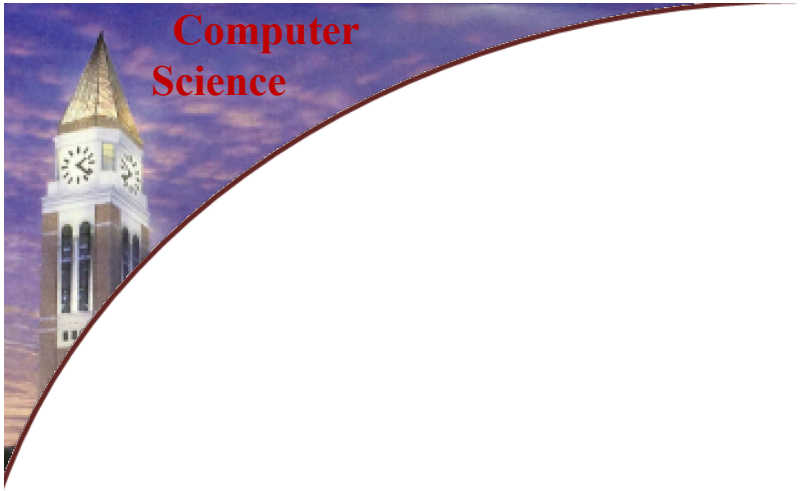


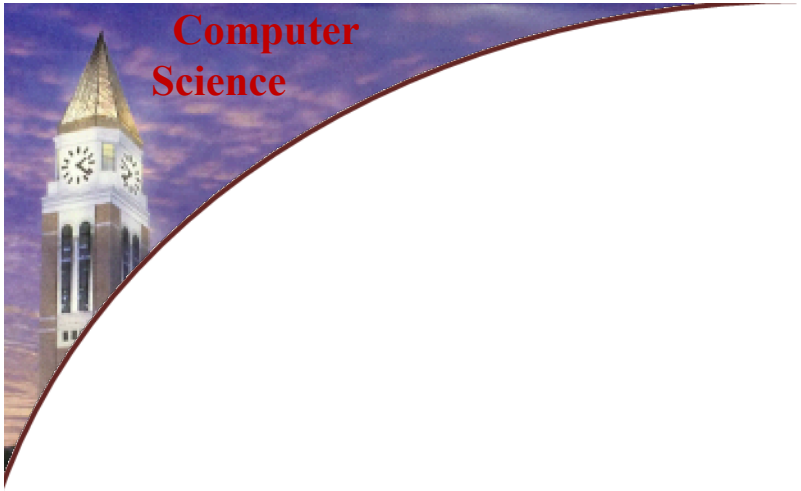
CSI 3350: PROGRAMMING LANGUAGES

Department of Computer Science & Engineering

Oakland University

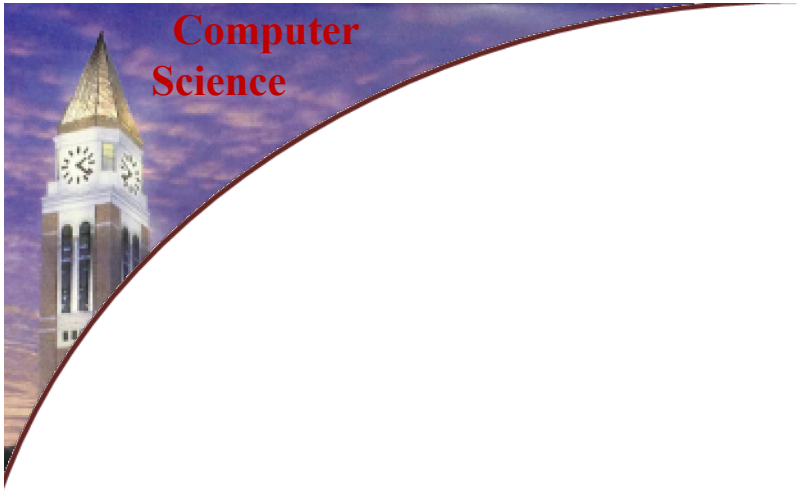



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```



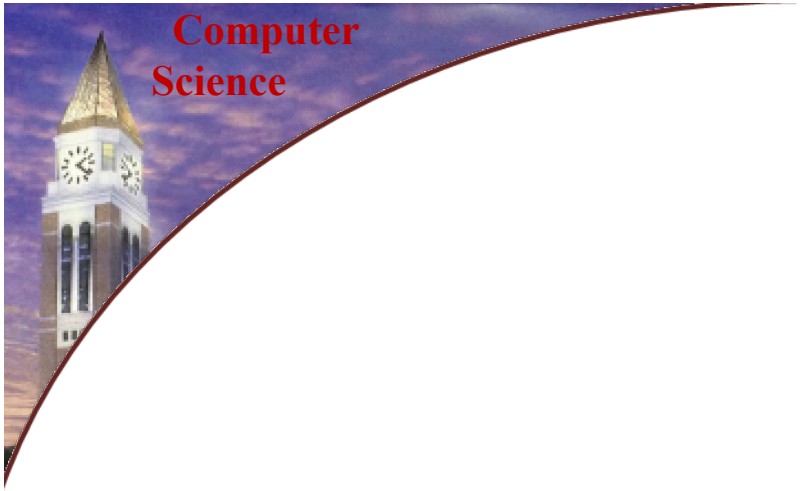

```
(fact 4)
```

```
(define      n = 4
     (fact n )
       (if
           (= n 0)
           1
           (* n (fact (- n 1)))
       )
)
```




```
(fact 4)
```

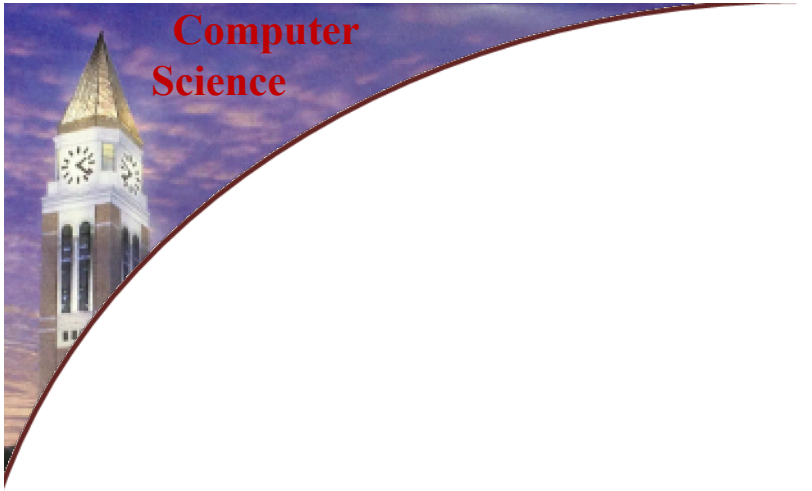
```
(define      n = 4
     (fact n )
       (if
           (= n 0)
           1
           (* n (fact (- n 1)))
       )
)
```




```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 4
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4 n - 1 = 3

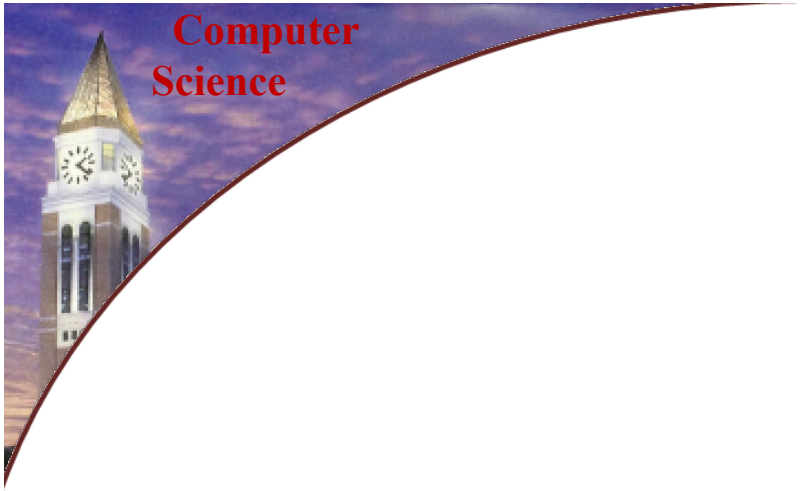



```
(fact 4)
= (* 4 (fact 3))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4

n - 1 = 3

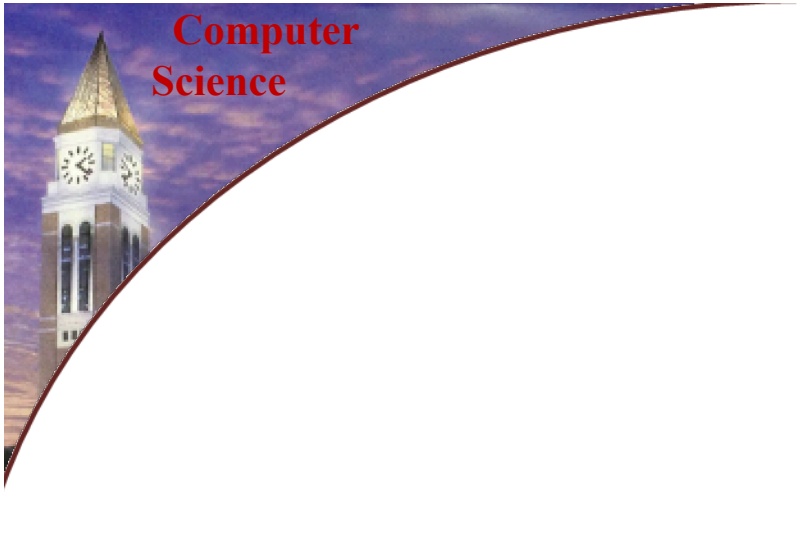



```
(fact 4)
= (* 4 (fact 3))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 4

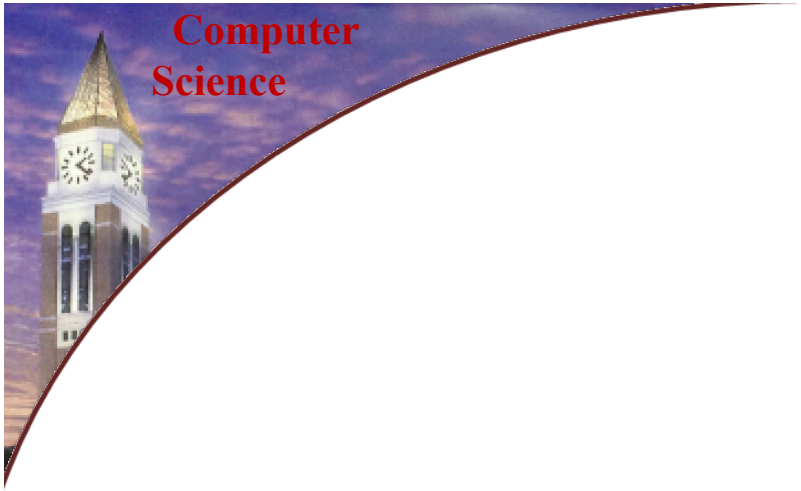
n - 1 = 3




```
(fact 4)
= (* 4 (fact 3))
```

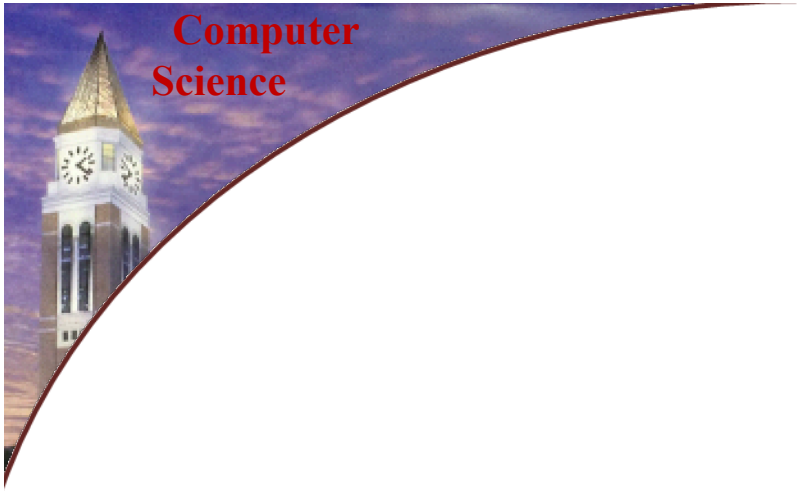
n = 3

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```



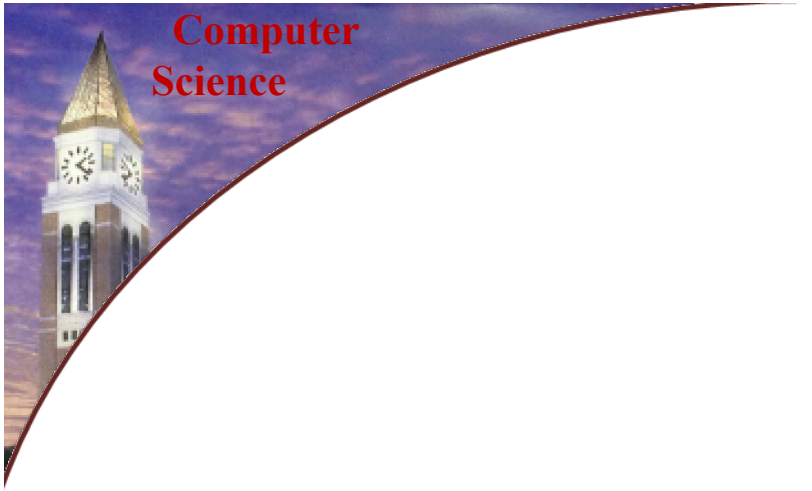

```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```




```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

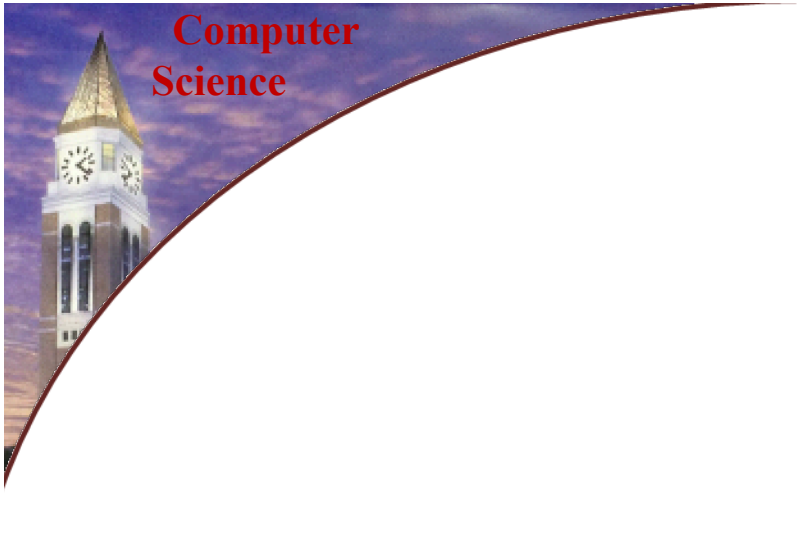
```
  (fact 4)
= (* 4 (fact 3))
```




```
(fact 4)
= (* 4 (fact 3))
```

```
(define      n = 3
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3 n - 1 = 2

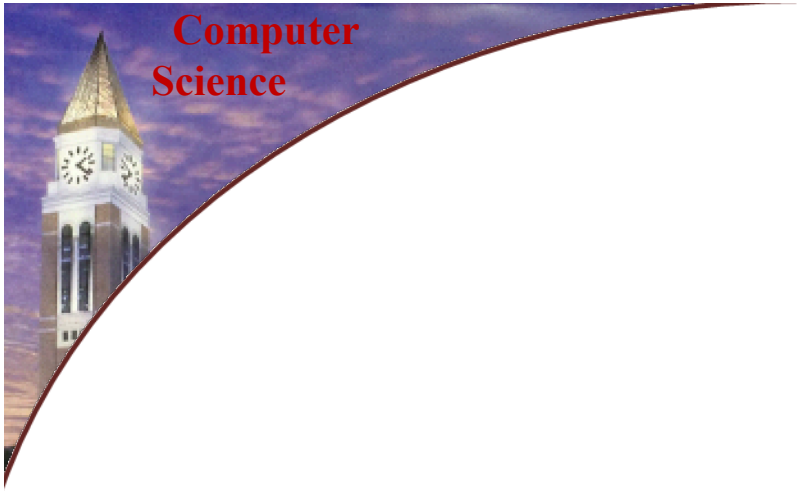



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

n = 3 **n - 1 = 2**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

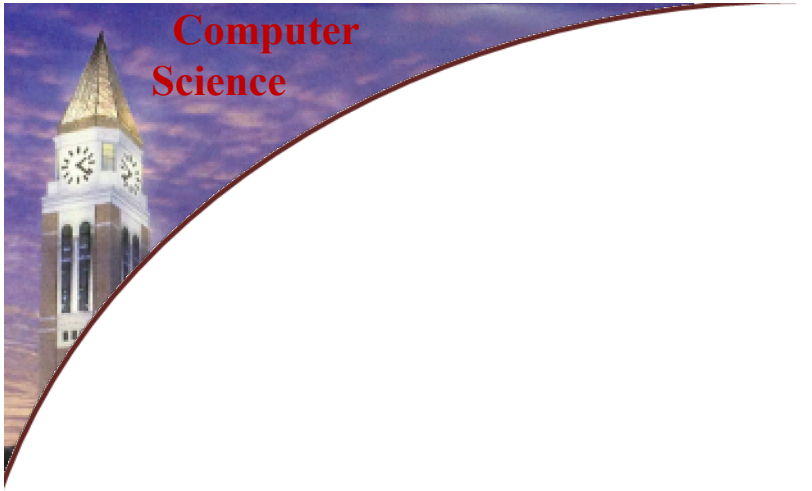



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

n = 3 **n - 1 = 2**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

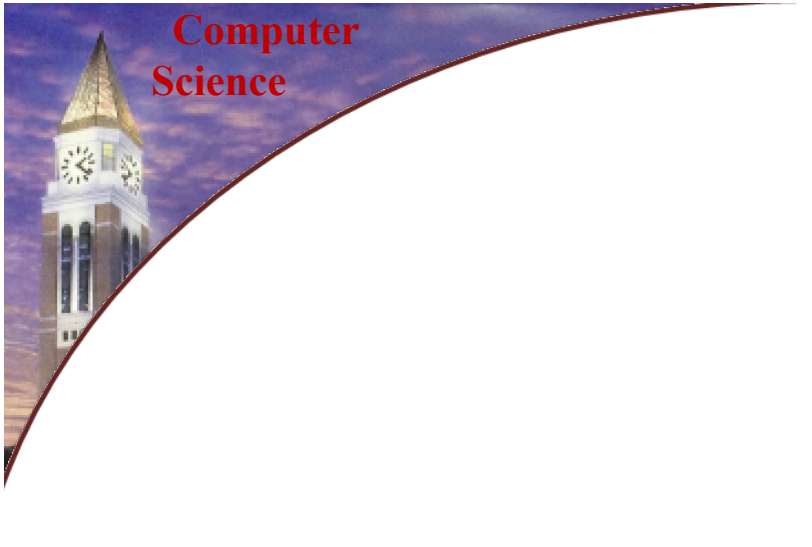



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

n - 1 = 2

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

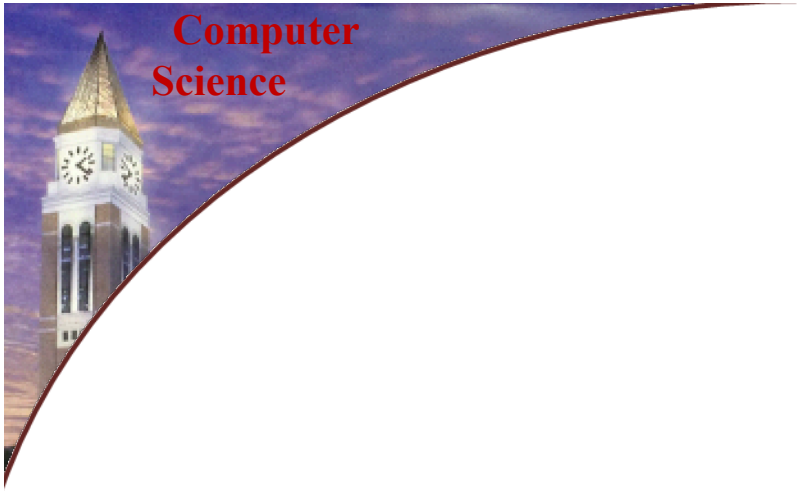



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 3

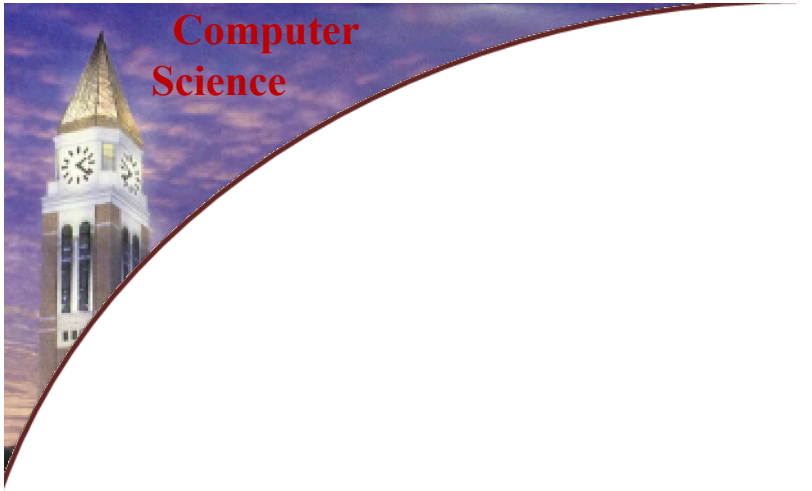
n - 1 = 2

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



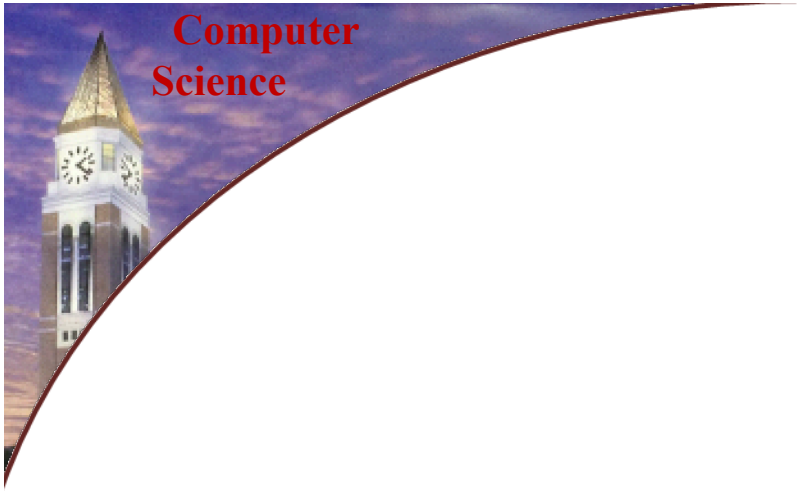

```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



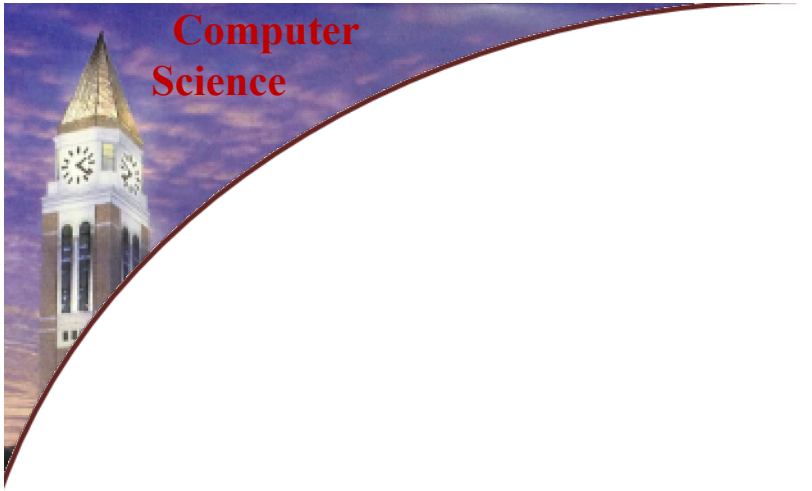

```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```




```
(define      n = 2
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

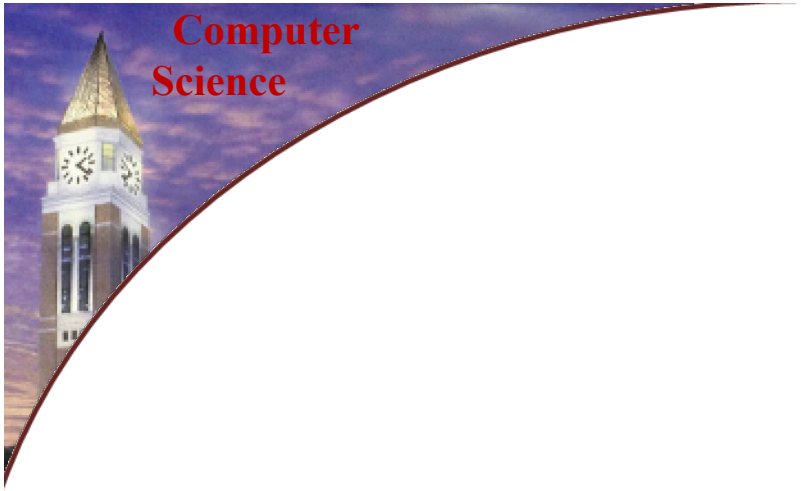



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n = 2 **n - 1 = 1**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```

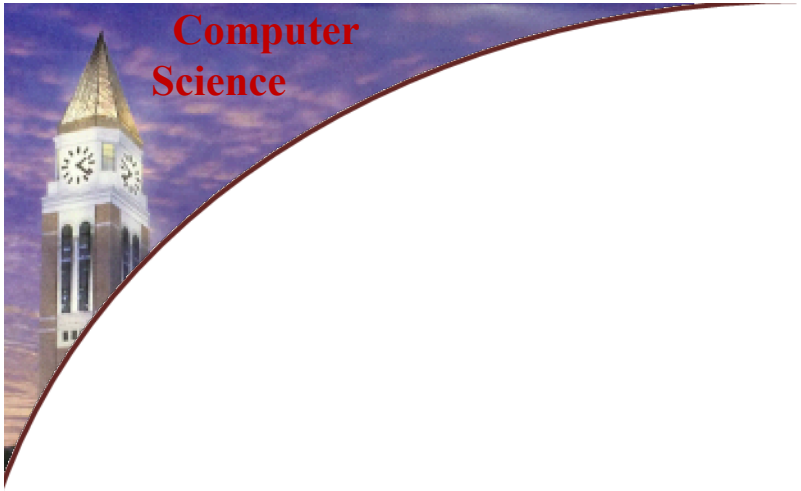



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n = 2 **n - 1 = 1**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

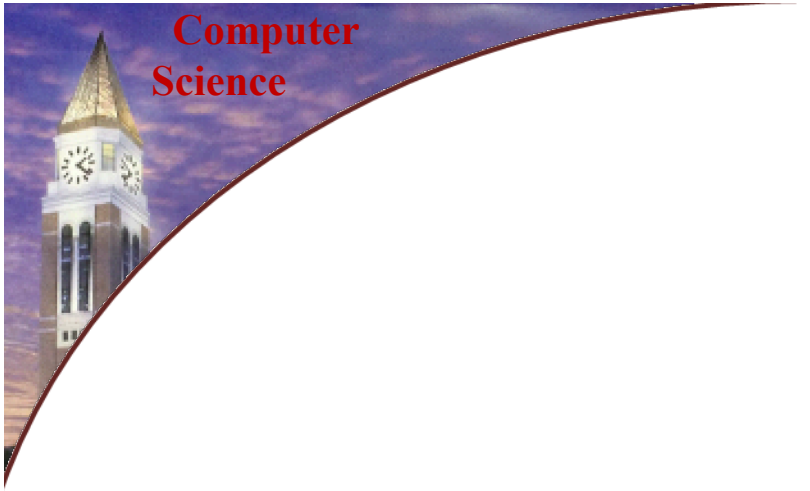



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n - 1 = 1

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

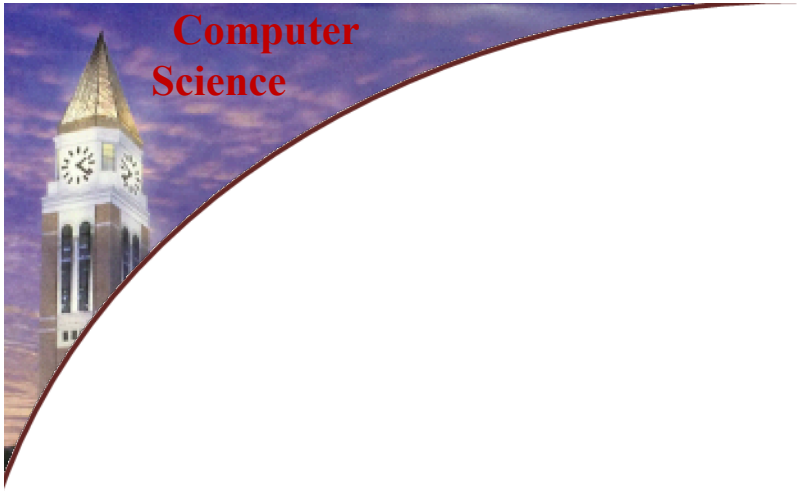



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 2

n - 1 = 1

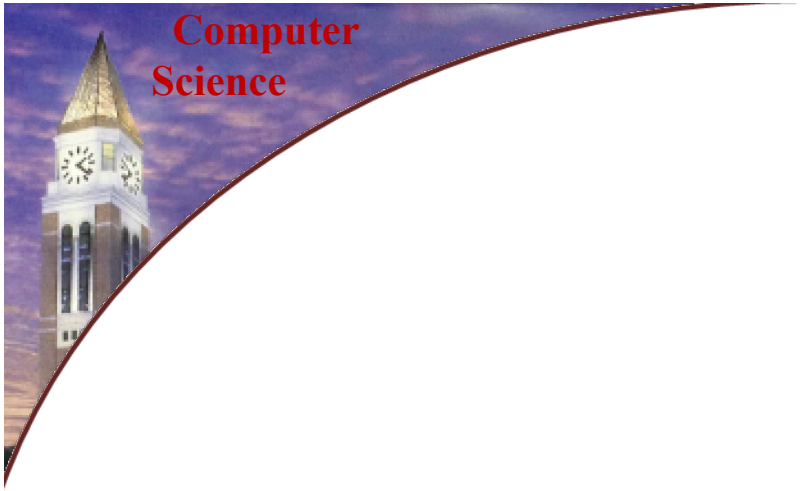
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

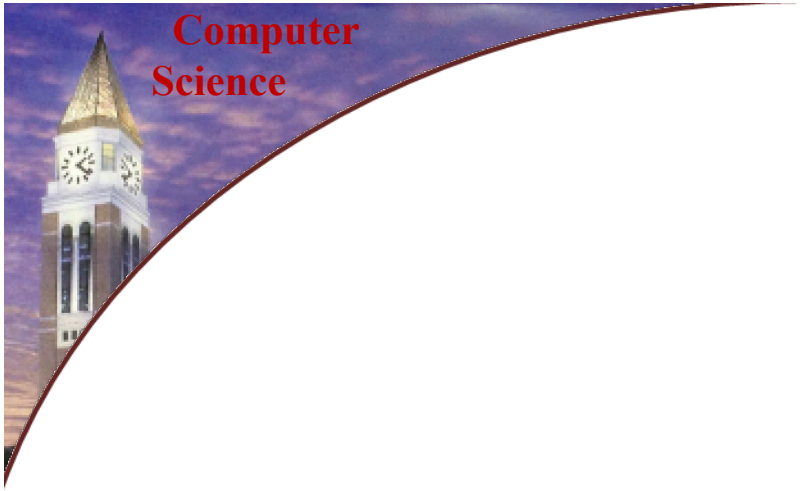
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

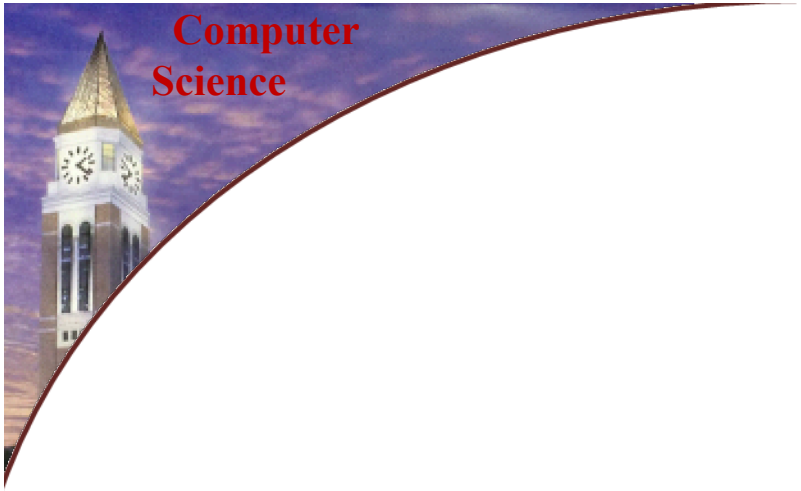



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n = 1 **n - 1 = 0**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```

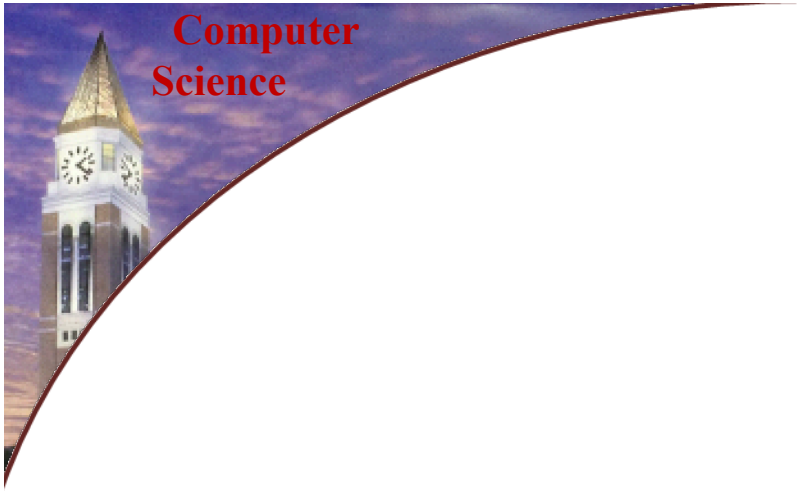



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n = 1 **n - 1 = 0**

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

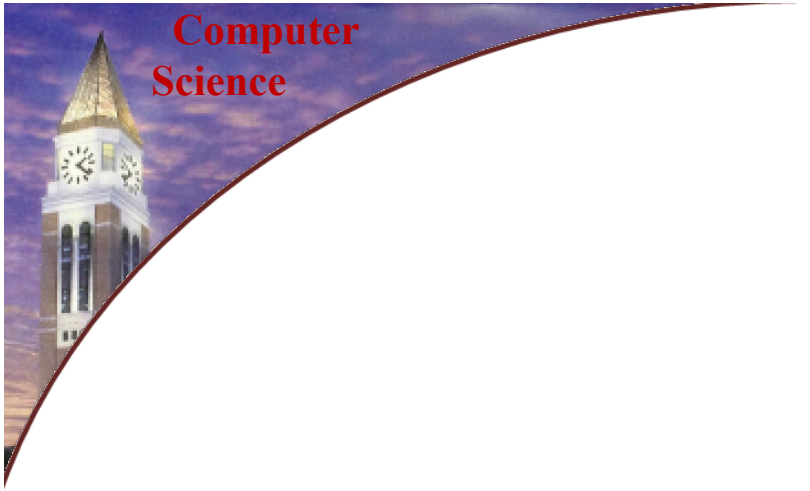



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n - 1 = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

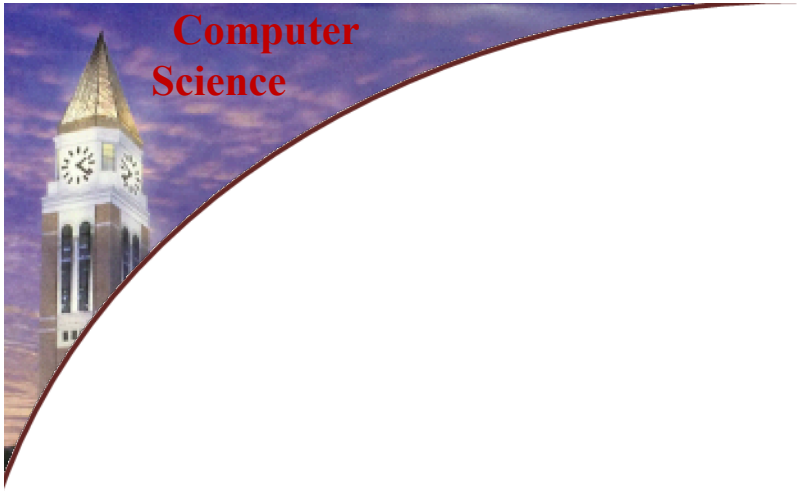



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 1

n - 1 = 0

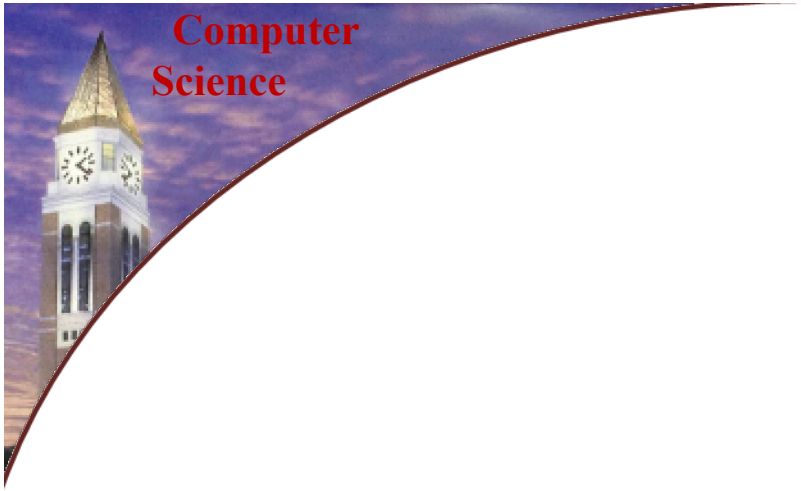
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 0

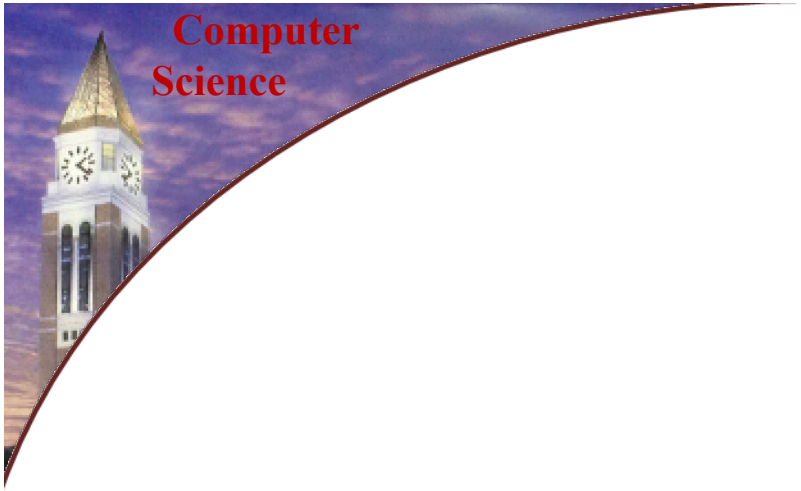
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```




```
(define
    (fact n )
      (if
        (= n 0)
        1
        (* n (fact (- n 1)))
      )
)
```

n = 0

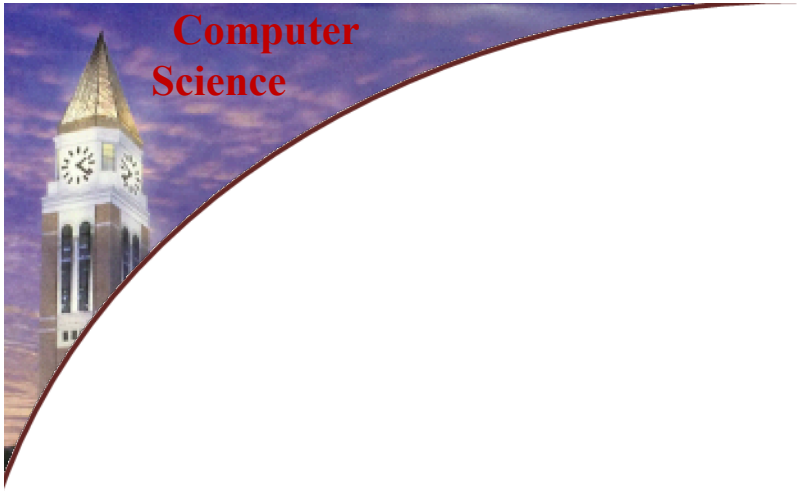
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 0

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```

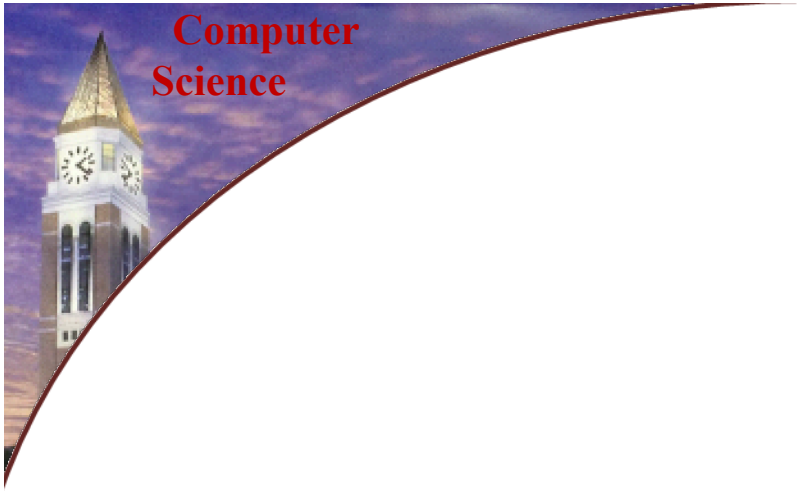



```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

n = 0

base case is hit !

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```




```
(define
    (fact n )
      (if
         (= n 0)
         1
         (* n (fact (- n 1)))
      )
)
```

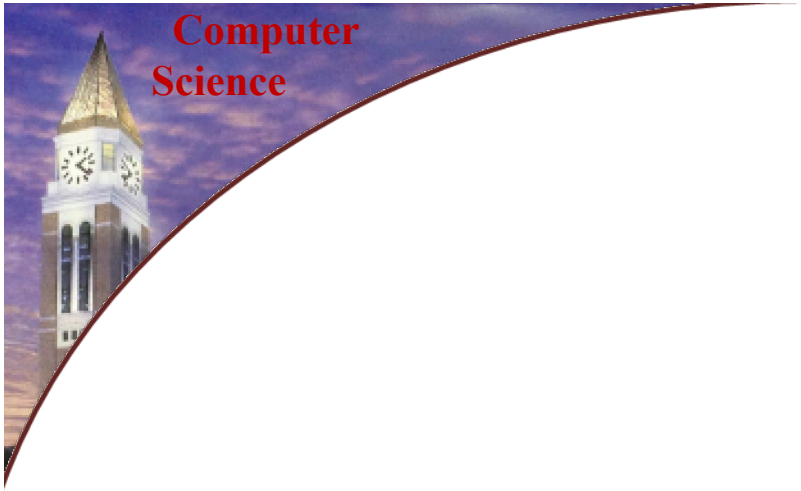
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
```

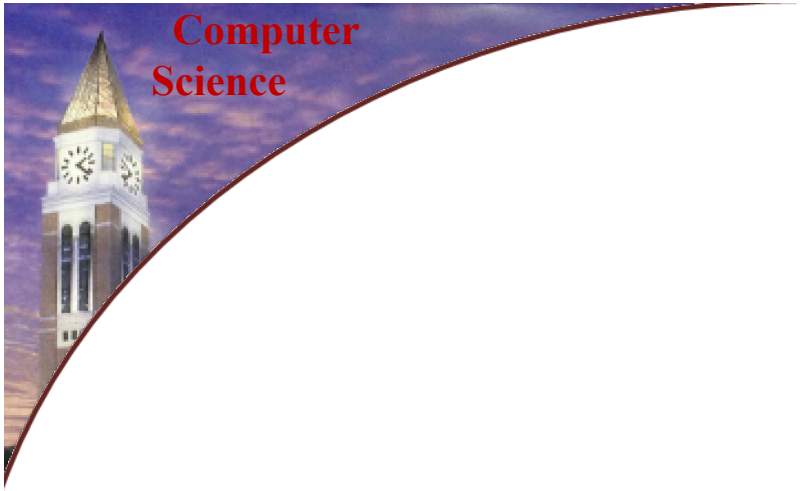
```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
```




```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

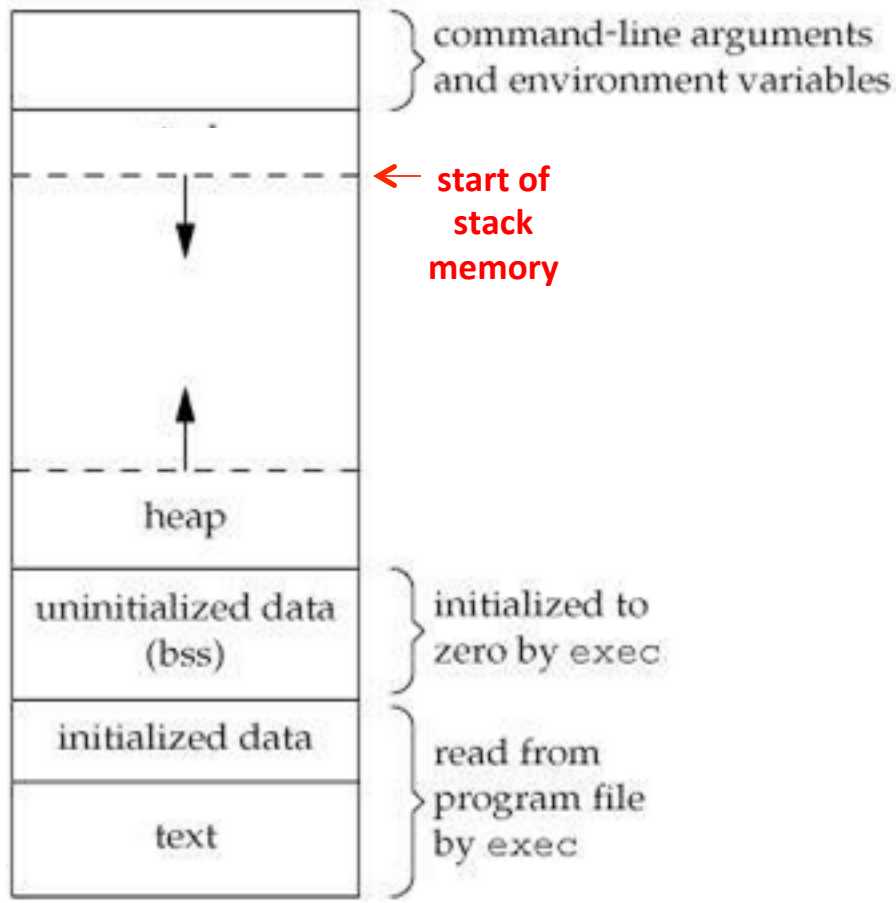
Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

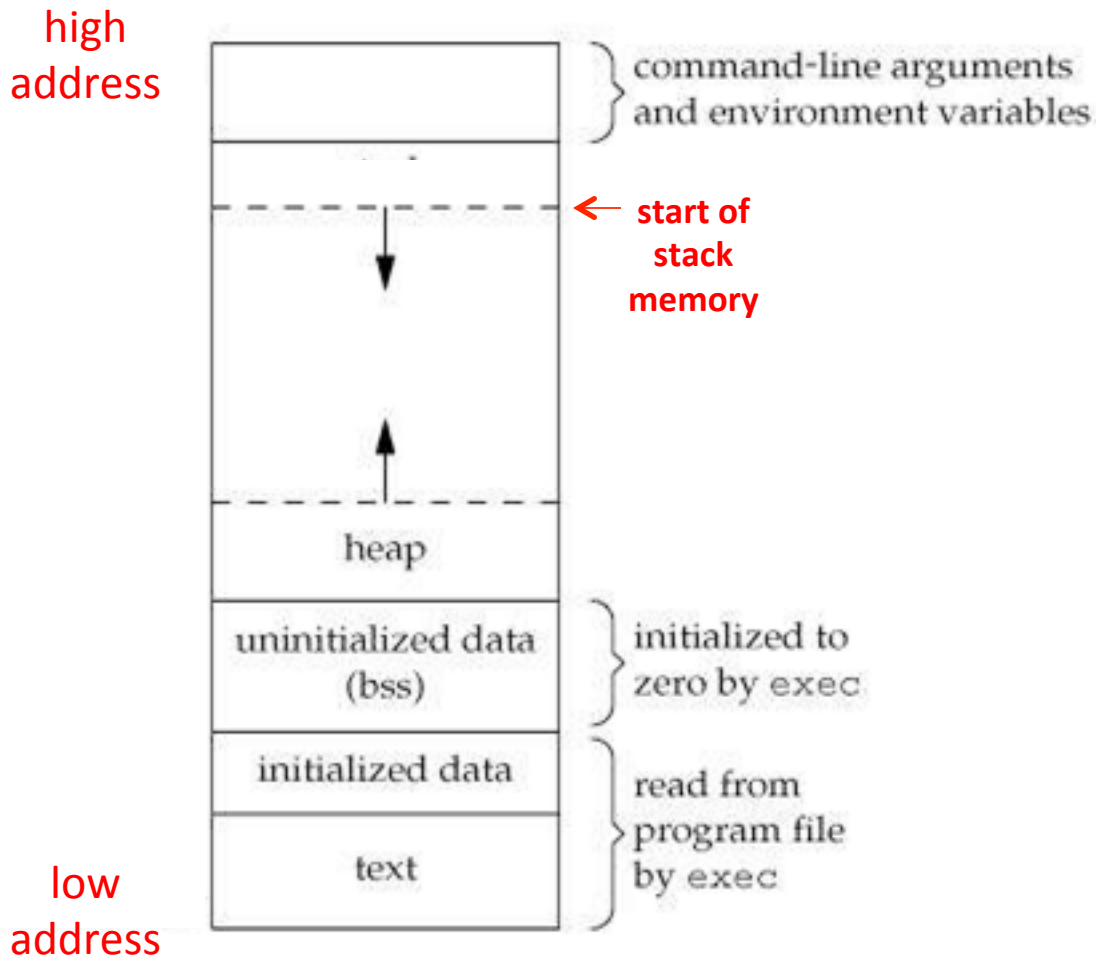
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```

Activation Record (AR)

The Main Memory

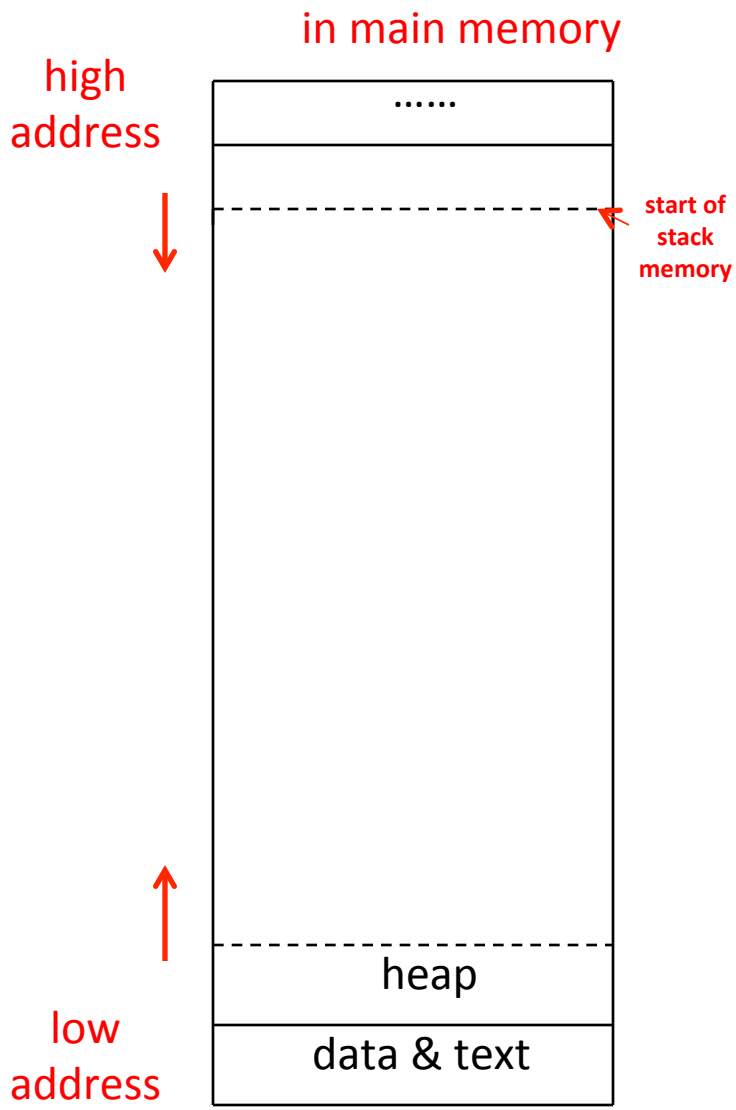


The Main Memory



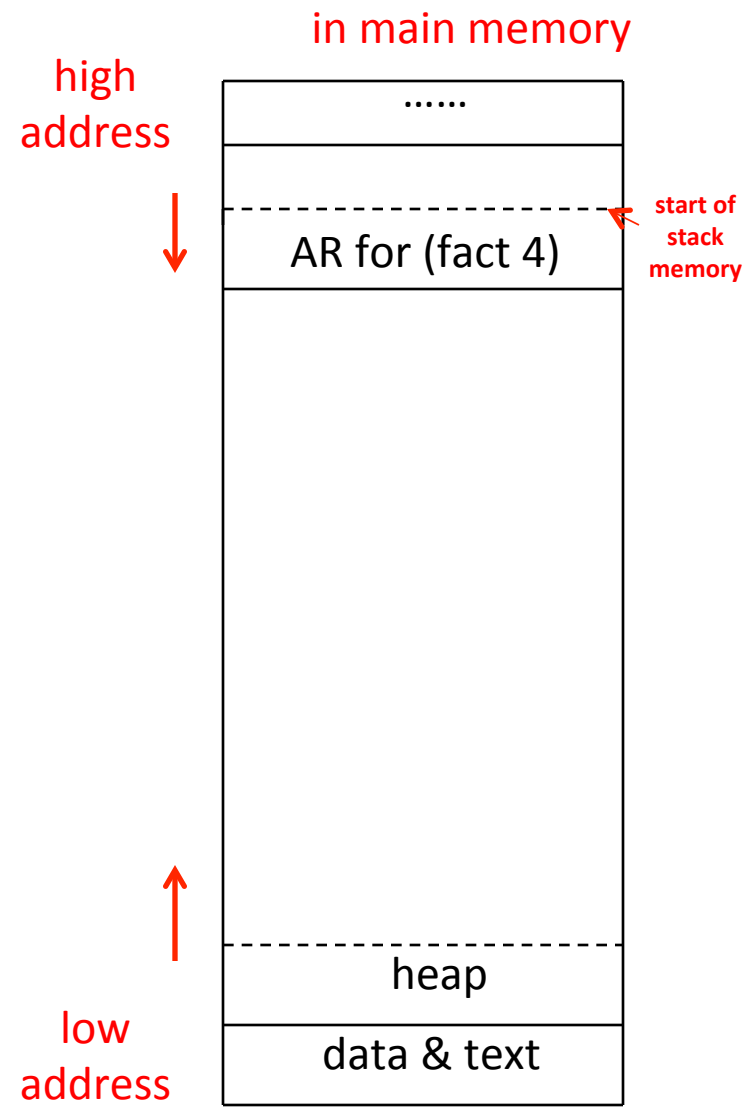
Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```



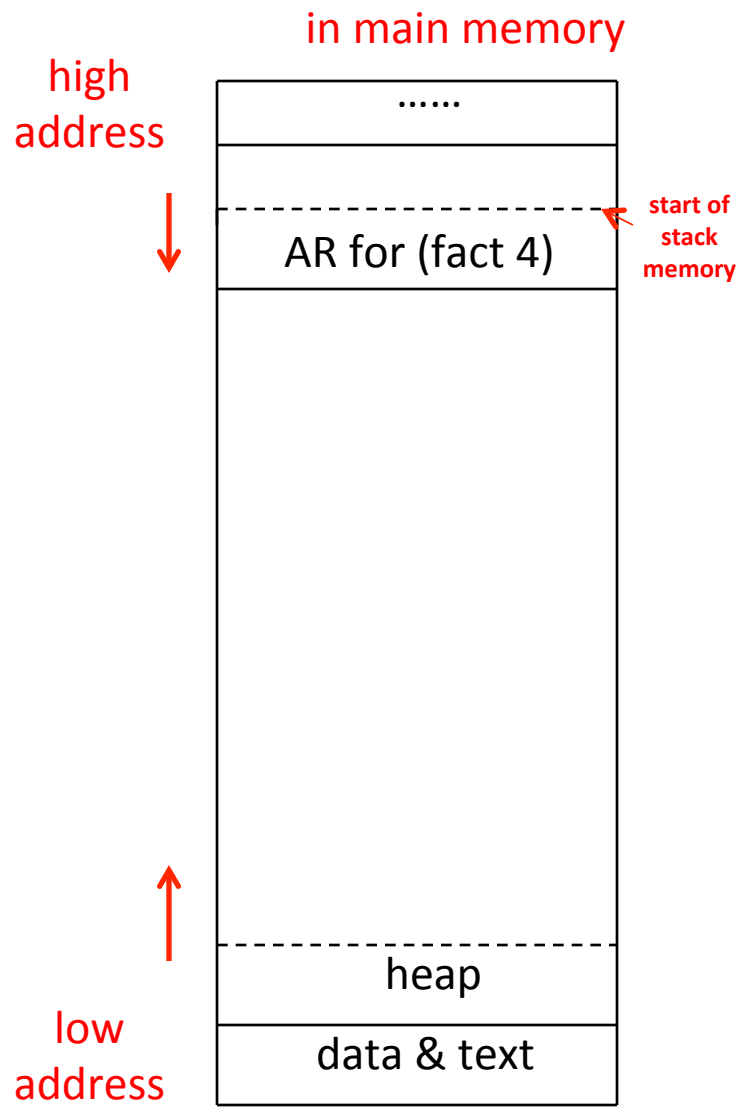
Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)

(fact 4)
```



Activation Record (AR)

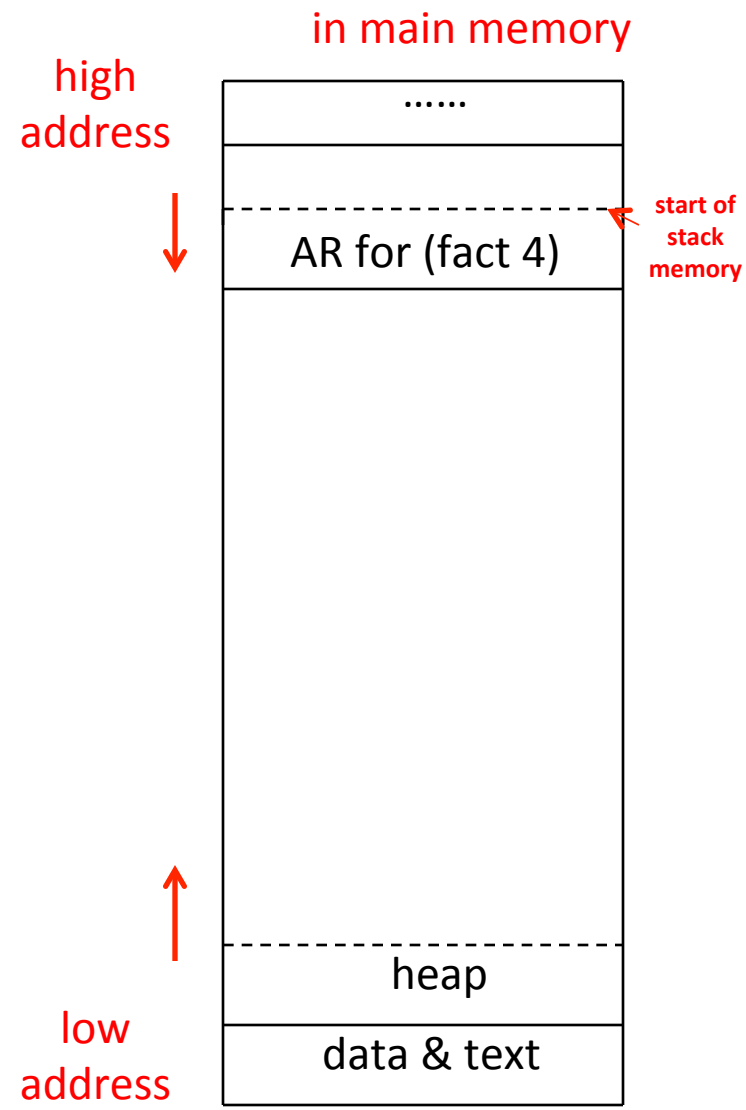
```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)

(fact 4)
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

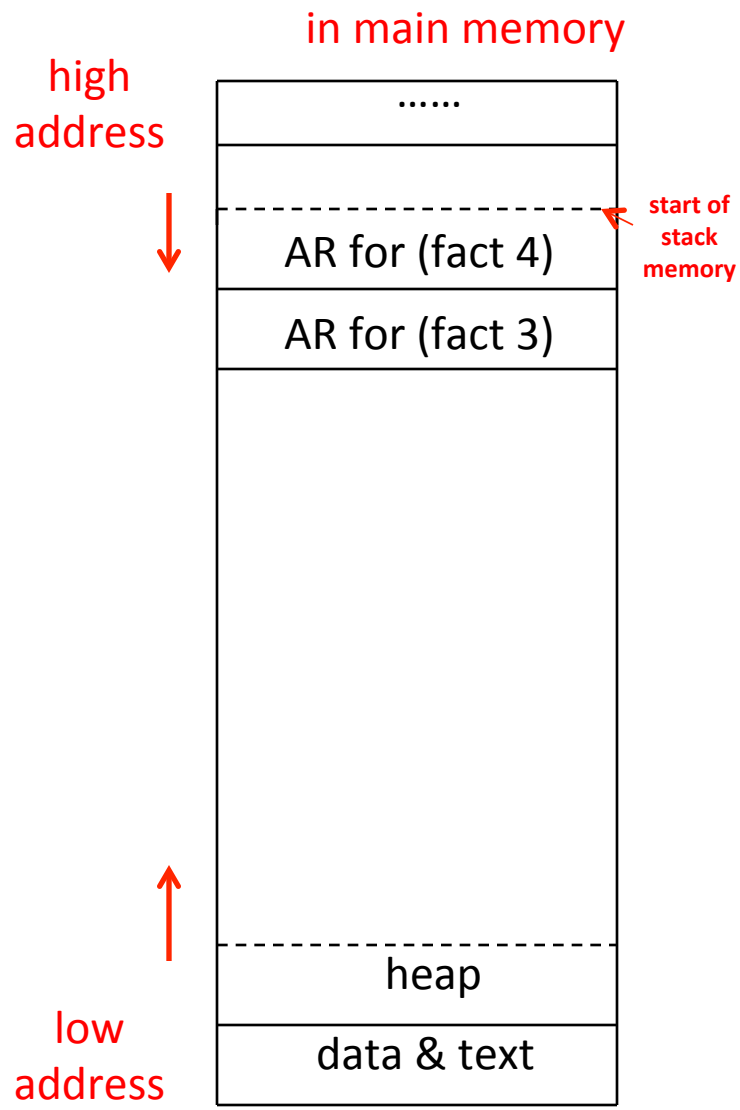
```
(fact 4)
= (* 4 (fact 3))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

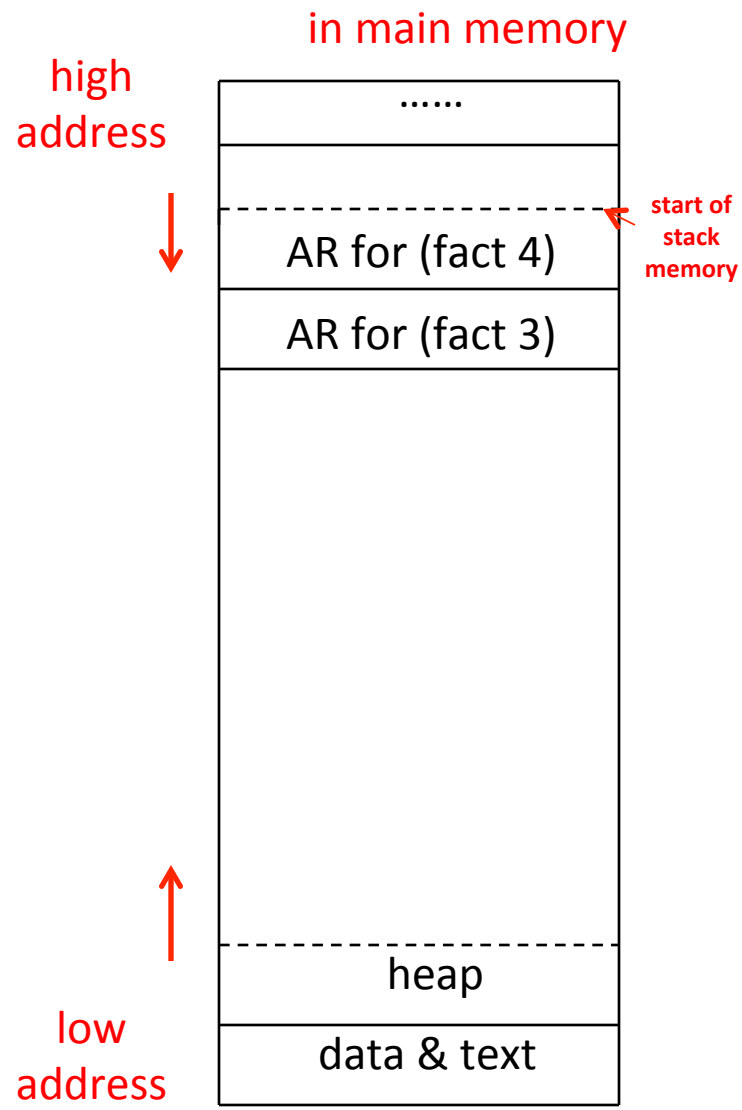
```
  (fact 4)
= (* 4 (fact 3))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

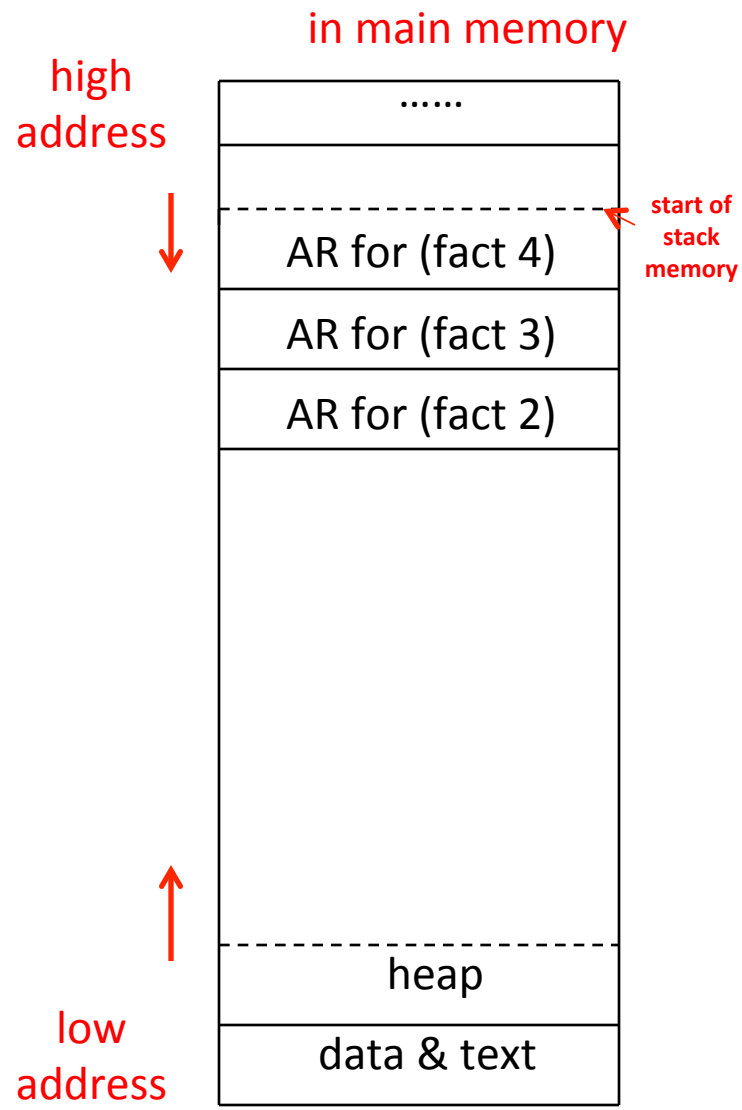
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

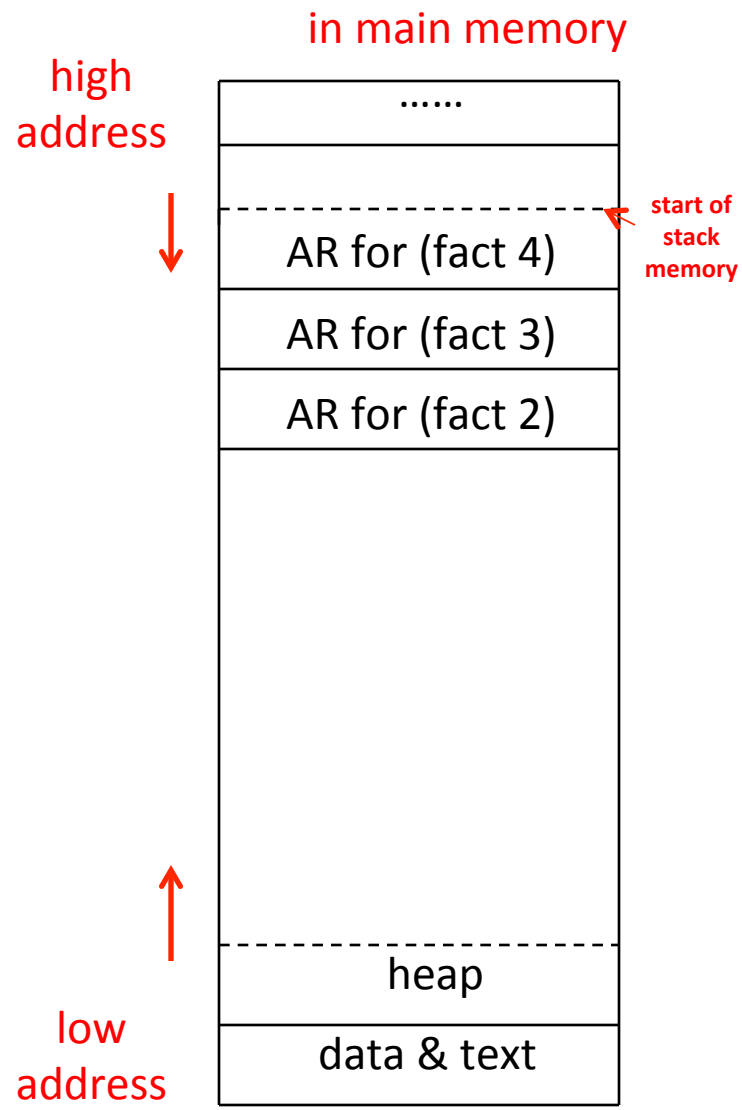
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

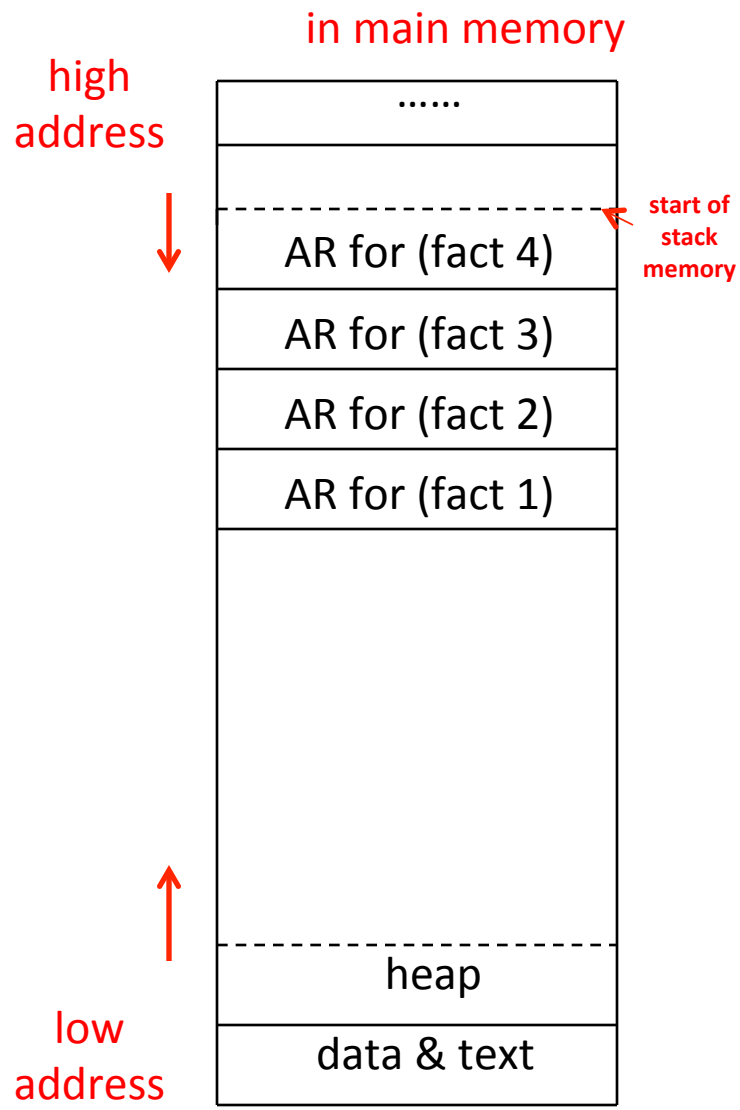
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

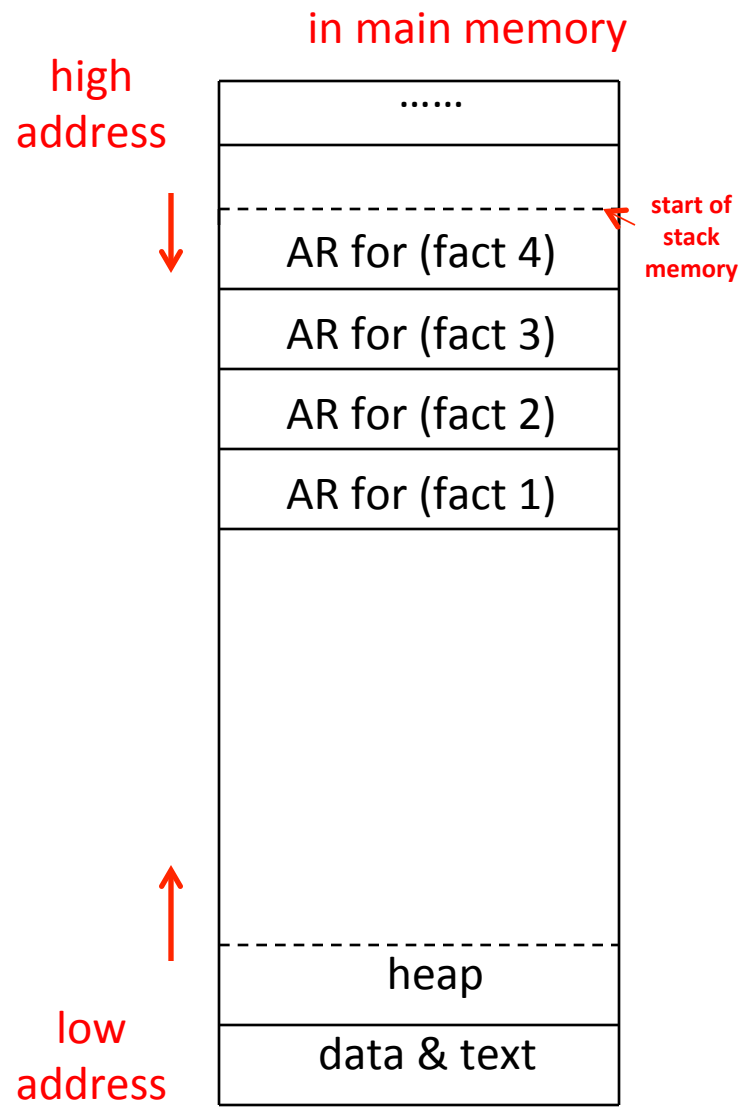
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

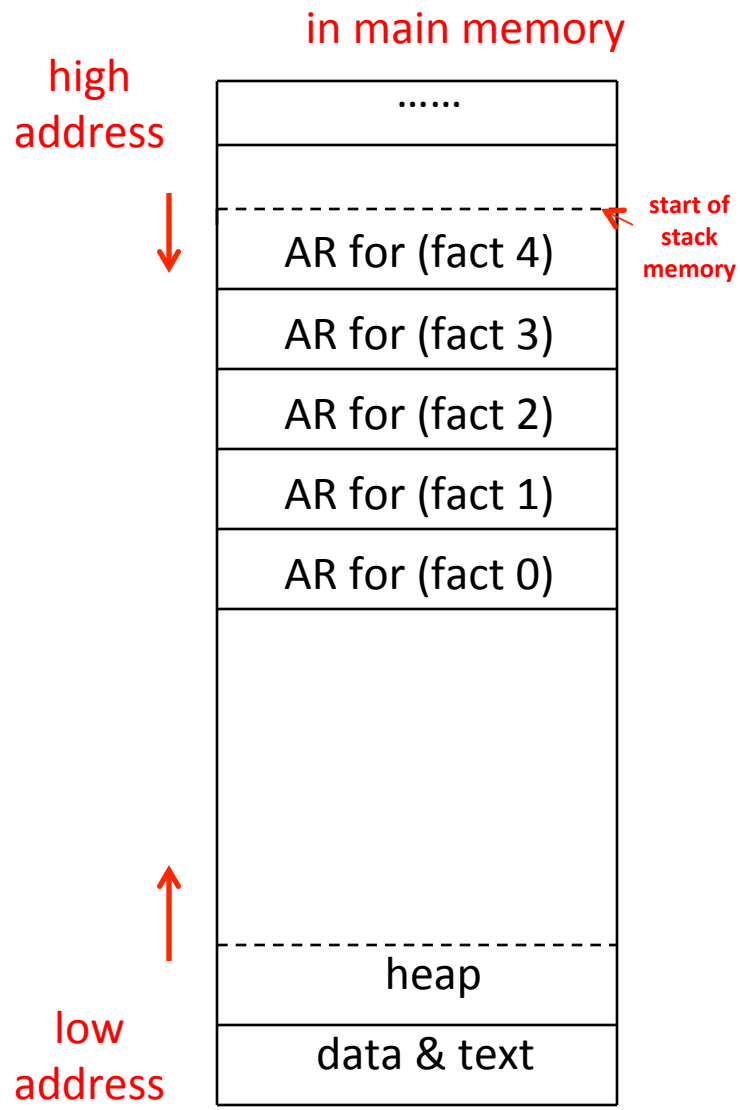
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

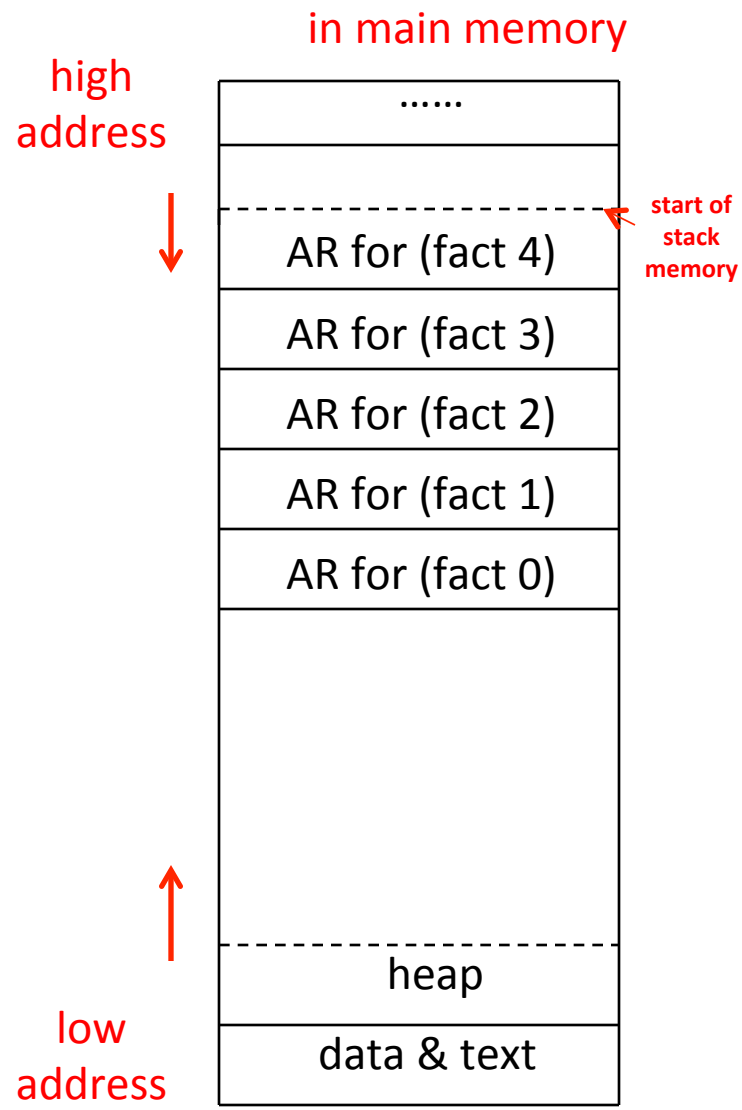
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

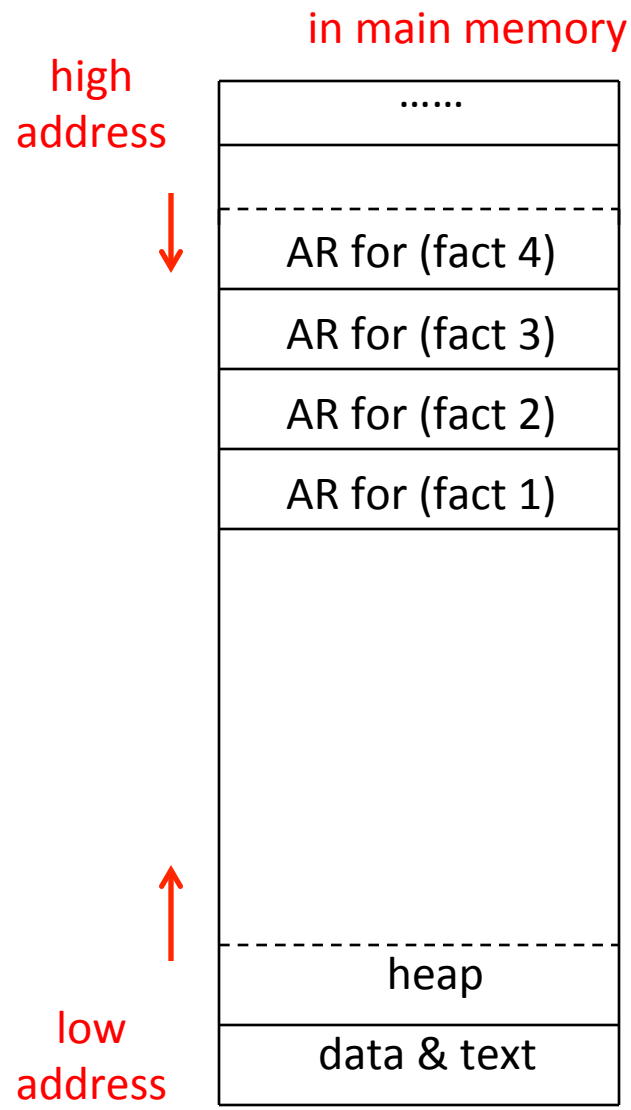
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

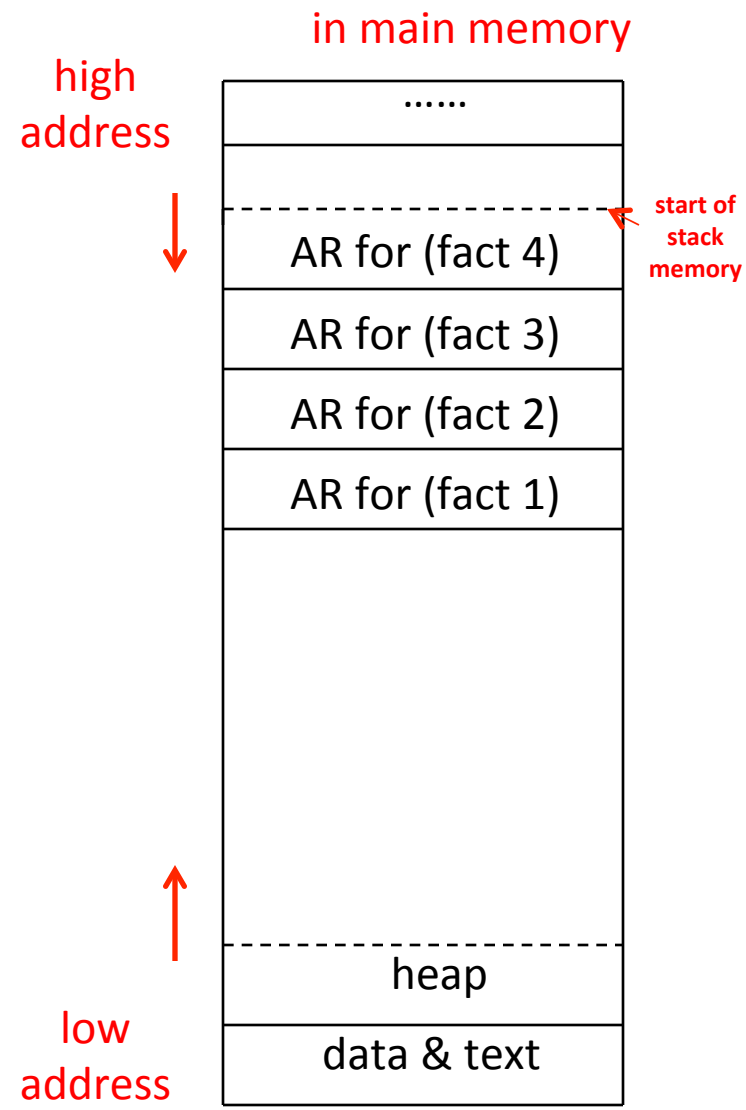
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

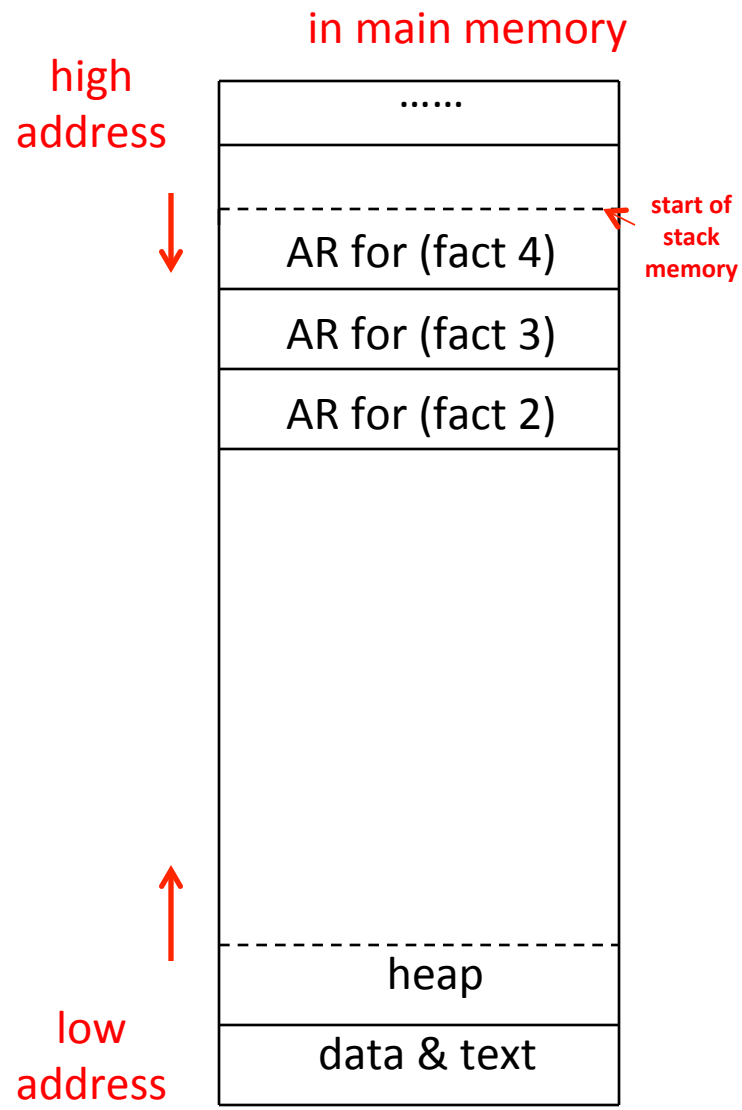
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

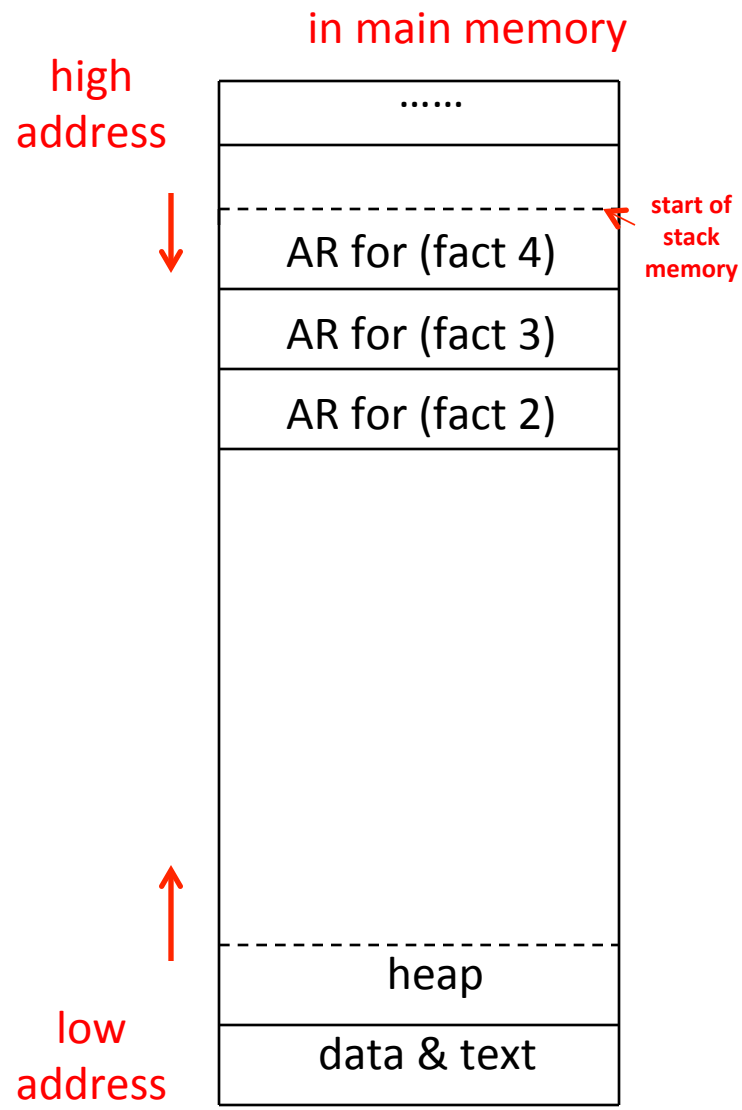
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

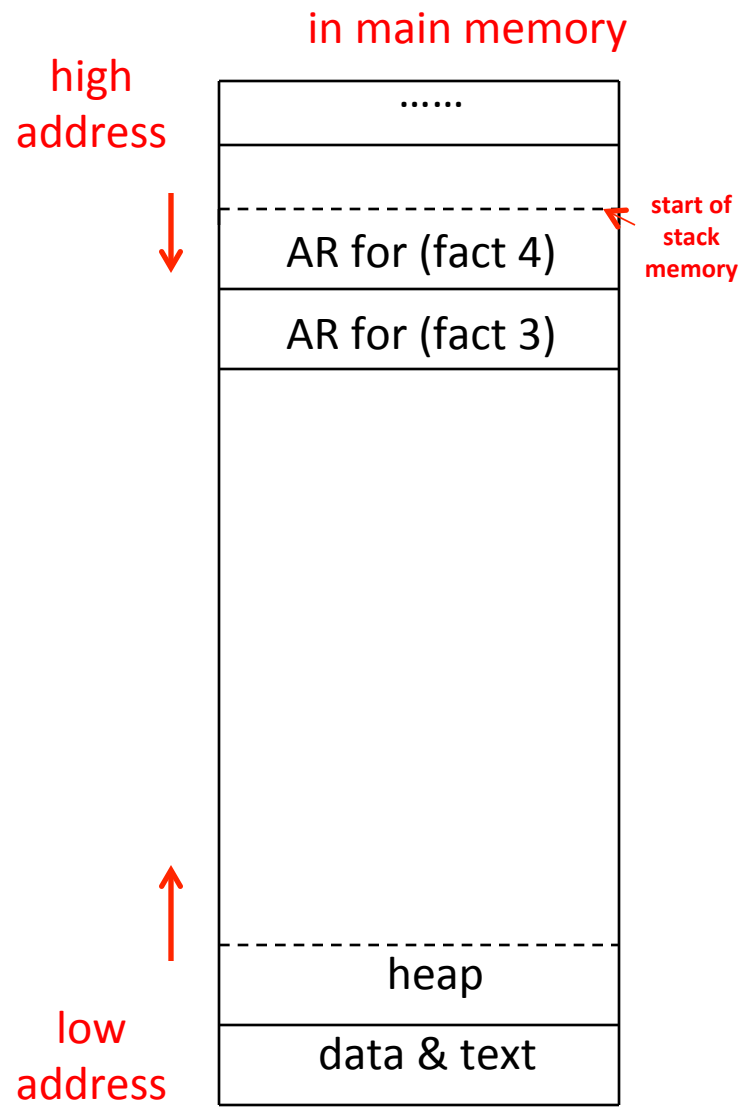
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

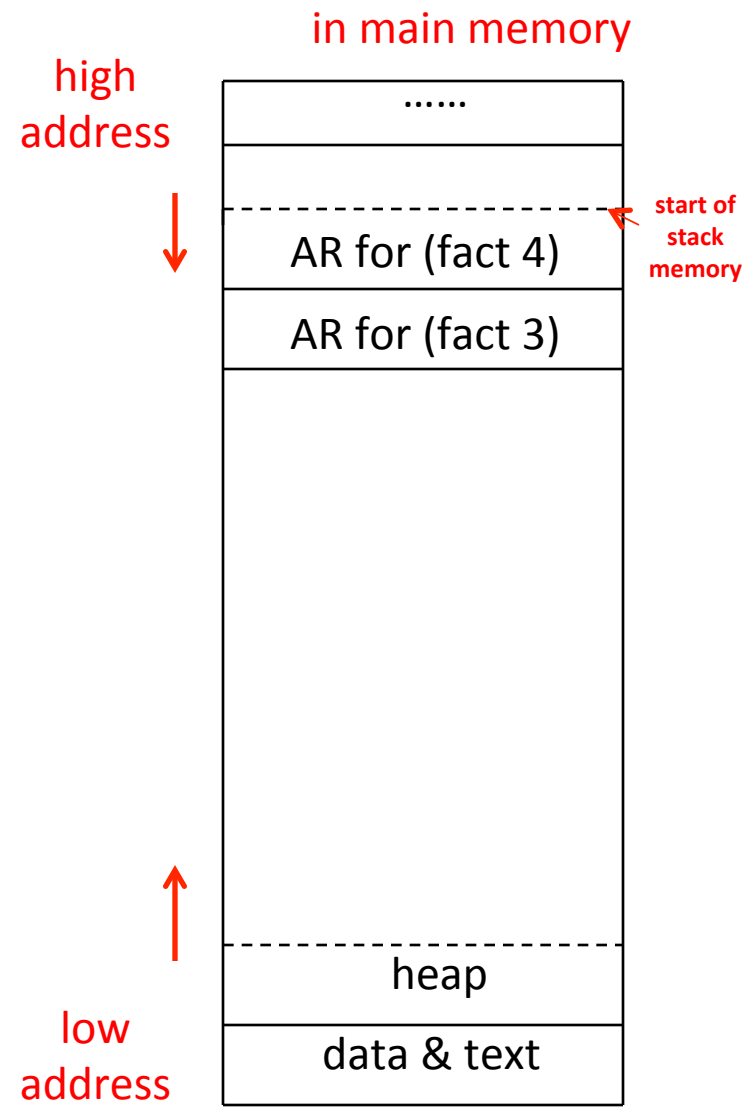
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

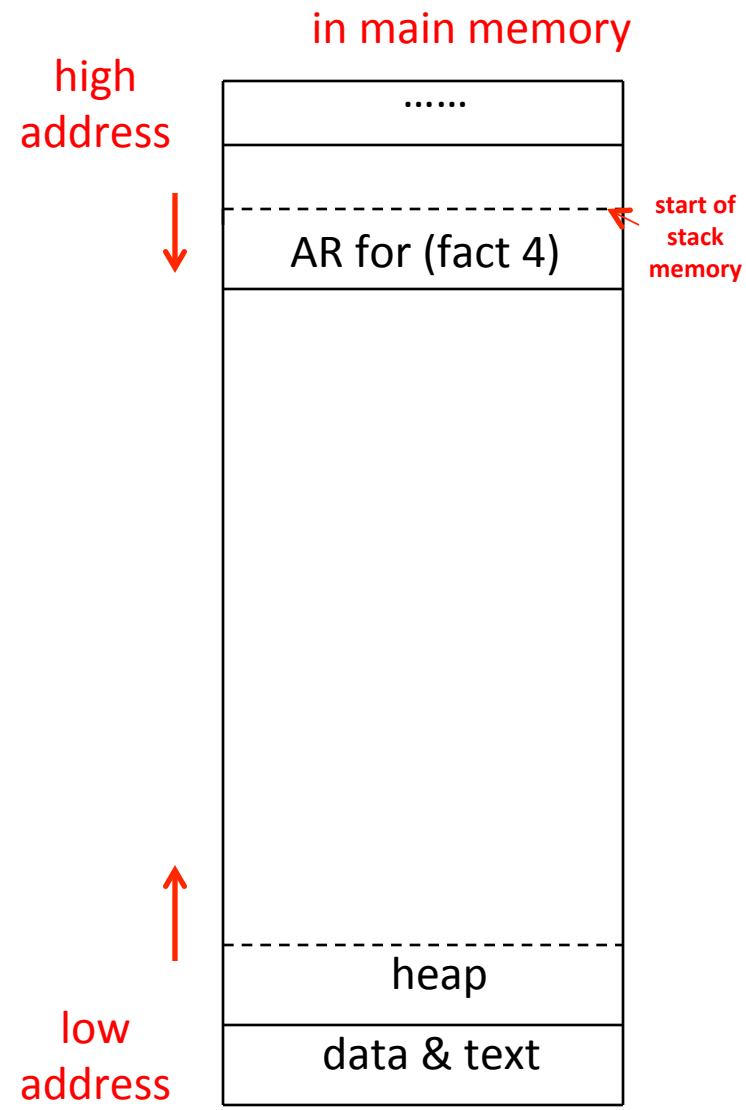
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
```



Activation Record (AR)

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1)))
      )
)
```

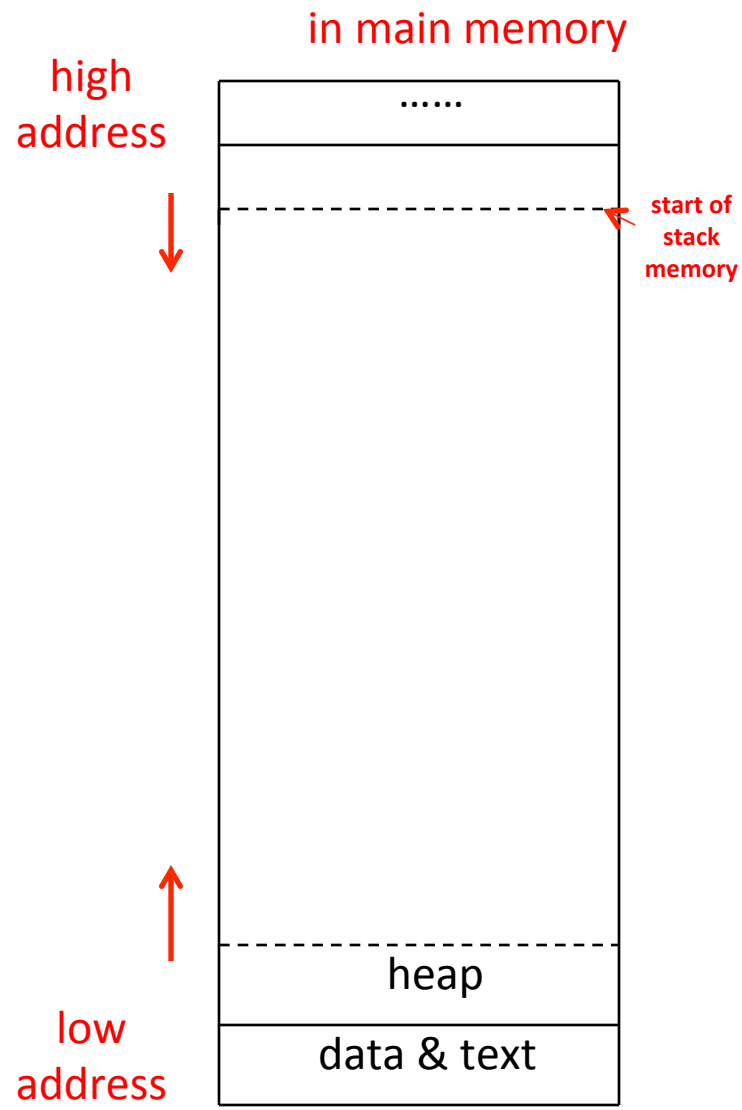
```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```



Activation Record (AR)

```
(define
    (fact n )
      (if
         (= n 0)
         1
         (* n (fact (- n 1)))
      )
)
```

```
  (fact 4)
= (* 4 (fact 3))
= (* 4 (* 3 (fact 2)))
= (* 4 (* 3 (* 2 (fact 1))))
= (* 4 (* 3 (* 2 (* 1 (fact 0)))))
= (* 4 (* 3 (* 2 (* 1 1))))
= (* 4 (* 3 (* 2 1)))
= (* 4 (* 3 2))
= (* 4 6)
= 24
```



```
(define
    (fact-new n )
    (fact-tail n 1)
)


(define
    (fact-tail n prod)
      (if
          (= n 0)
          prod
          (fact-tail (- n 1) (* n prod) )
      )
)
```

tail recursion!

```
(define
    (fact n )
      (if
          (= n 0)
          1
          (* n (fact (- n 1))))
)
```

Pairwise Reversal of A List

- `(define (pairwise-reversal lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(2 1 3)
'( 1 2 3 4)   => '(2 1 4 3)
```

- `(define (pairwise-reversal-odd lst) ... )`

```
'(1)   => '(1)
'( 1 2 3)   => '(1 3 2)
'( 1 2 3 4)   => '(2 1 4 3)
```

Pairwise Reversal of A List

- **(define (pairwise-reversal lst) ...)**

```
'(1)   => '(1)
'( 1 2 3)   => '(2 1 3)
'( 1 2 3 4)   => '(2 1 4 3)
```

if **pairwise-reversal** is given to you as a library function, how to code **pairwise-reversal-odd** ?

- **(define (pairwise-reversal-odd lst) ...)**

```
'(1)   => '(1)
'( 1 2 3)   => '(1 3 2)
'( 1 2 3 4)   => '(2 1 4 3)
```

Pairwise Reversal of A List

- **(define (pairwise-reversal lst) ...)**

```
'(1)   => '(1)
'( 1 2 3)   => '(2 1 3)
'( 1 2 3 4)   => '(2 1 4 3)
```

if **pairwise-reversal** is given to you as a library function, how to code **pairwise-reversal-odd** ?

- **(define (pairwise-reversal-odd lst) ...)**

```
'(1)   => '(1)
'( 1 2 3)   => '(1 3 2)
'( 1 2 3 4)   => '(2 1 4 3)
```

Anonymous Functions

- F(x) = (x + 1)
- g(x) = (x + 1)

Lambda Expression

- $F(x) = (x + 1)$

function name

binding variable

function body

function definition

```
(lambda (x) (+ x 1))
```

A Function of Two Forms of Definition

- $F(x) = (x + 1)$

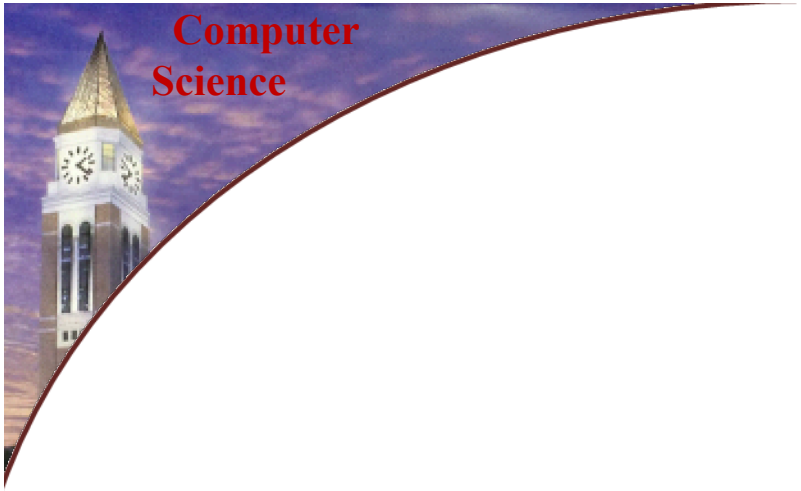
```
(define ( F x )
     (+ x 1)
)
```

```
(define F
     ( lambda (x) (+ x 1) )
)
```

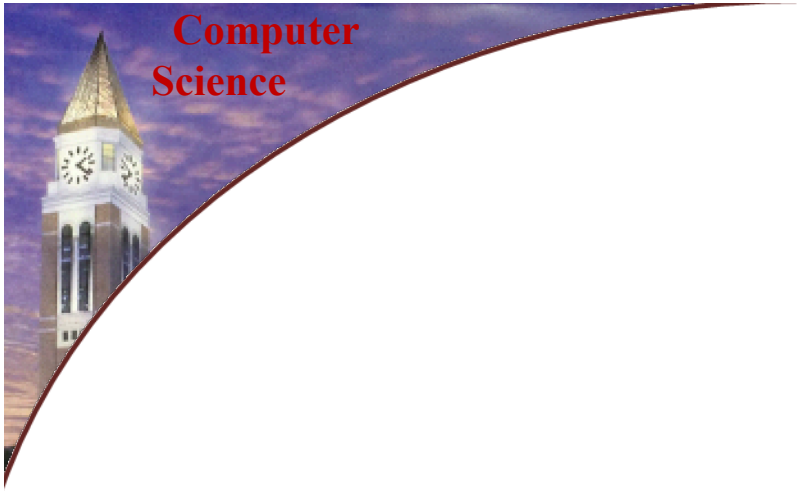
- $G(x) = (x + 1)$

```
(define ( G x )
     (+ x 1)
)
```

```
(define G F)
```



HW02



HW02

Read the test file first !

Carpet

(define (carpet n) ...)

```
n = 0    `(  (%)  )

n = 1    `(  (+ + +)
             (+ % +)
             (+ + +)  )
                      (carpet 0)

n = 2    `(  (% % % % %)
             (% + + + %)
             (% + % + %)
             (% + + + %)
             (% % % % %)   )
                      (carpet 1)

n = 3    `(  (+ + + + + + +)
             (+ % % % % % +)
             (+ % + + + % +)
             (+ % + % + % +)
             (+ % + + + % +)
             (+ % % % % % +)
             (+ + + + + + +)  )
                      (carpet 2)
```

Carpet

```
(carpet 3)
`(  (+ + + + + + +)
    (+ % % % % % +)
    (+ % + + + % +)
    (+ % + % + % +)
    (+ % + + + % +)
    (+ % % % % % +)
    (+ + + + + + +)  )
```

(carpet 4) =
step-1: for-each list in `(carpet 3)` expand it by adding `'%` to the beginning and end of it

```
`(  (% + + + + + + + %)
    (% + % % % % % + %)
    (% + % + + + % + %)
    (% + % + % + % + %)
    (% + % + + + % + %)
    (% + % % % % % + %)
    (% + + + + + + + %))
```

step-2: add `'(%%%%%%%%%%)` to the beginning and the end of the result returned by step-1

```
(carpet 4)
`(  (% % % % % % % % %)
    (% + + + + + + + %)
    (% + % % % % % + %)
    (% + % + + + % + %)
    (% + % + % + % + %)
    (% + % + + + % + %)
    (% % % % % % % % %))
```

Pascal Triangle

(define (pascal n) ...)

```
n = 1    `( (1) )

n = 2    `( (1)
            (1 1) )

n = 3    `( (1)
            (1 1)
            (1 2 1)  )
```

<-> `((1) (1 1))

<-> `((1) (1 1) (1 2 1))

(pascal 4) = inserting '(1 3 3 1) to the end of (pascal 3)

n = 3

```
n = 4    `( (1)
            (1 1)
            (1 2 1)
            (1 3 3 1) )
```

<-> `((1) (1 1) (1 2 1) (1 3 3 1))

Pascal Triangle

(define (pascal n) ...)

```
n = 3    `( (1)
            (1 1)
            (1 2 1)  )
```

```
n = 4    `( (1)
            (1 1)
            (1 2 1)
            (1 3 3 1) )
```

get the last element from (pascal 3)

↓

generate `(1 3 3 1)

↓

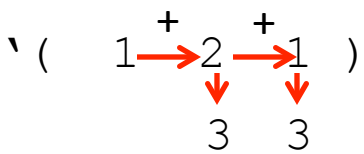
insert it into (pascal 3)

☺

(pascal 4) = inserting '(1 3 3 1) to the end of (pascal 3)

Pascal Triangle

- Generate `'(1 3 3 1)` from `'(1 2 1)`



Pascal Triangle

- Generate `'(1 3 3 1)` from `'(1 2 1)`

```
'(   1 --+--> 2 --+--> 1  )
         |        |
         v        v
     '  ( 3      3 )
```

We first generate the sub-list `'(3 3)`

Pascal Triangle

- Generate `'(1 3 3 1)` from `'(1 2 1)`

```
                +    +
'(   1 ---> 2 ---> 1  )
            |      |
            v      v
'(   1   3   3   1  )
     ^               ^
```

At the very end, we just insert these two `1`'s into the beginning and end of `'(3 3)`

Pascal Triangle

(define (pascal n) ...)

```
n = 3    `( (1)
            (1 1)
            (1 2 1)  )
```

```
n = 4    `( (1)
            (1 1)
            (1 2 1)
            (1 3 3 1) )
```

get the last element from (pascal 3)

↓

generate `(1 3 3 1)

↓

insert it into (pascal 3)

☺

(pascal 4) = inserting '(1 3 3 1) to the end of (pascal 3)

Create-mapping

```
`(I  II  III  IV  V  )    key-list

`(1   2   3   4   5  )    value-list
```

Search-key: `IV`

Create-mapping

```
`(I  II  III  IV  V  )    key-list

`(1   2   3   4   5  )    value-list
```

Search-key: `IV`

Create-mapping

```
`(II  III  IV  V   )    key-list

`(2   3    4   5   )    value-list
```

Search-key: `IV`

Create-mapping

```
'(II  III  IV  V   )    key-list

'(2   3    4   5   )    value-list
```

Search-key: `IV`

Create-mapping

```
`(III IV V  )    key-list

`(3   4   5  )   value-list
```

Search-key: `IV`

Create-mapping

```
`(III IV V  )    key-list
 `(3   4  5  )    value-list
```

Search-key: `IV`

Create-mapping

```
`(IV V  )    key-list
`(4  5  )    value-list
```

Search-key: `IV`

Create-mapping

```
`(IV V  )    key-list
`(4  5  )    value-list
```

Search-key: `IV`