Garter
Garbage Collection
Garter / GC

Example 1
let x = (1, 2), y = let tmp = (10, 20) in tmp[0] + tmp[1], p0 = x[0] + y, p1 = x[1] + y in (p0, p1)
let \( x = (1, 2) \), \( y = \) let \( \text{tmp} = (10, 20) \) in \( \text{tmp}[0] + \text{tmp}[1] \), \( p0 = x[0] + y \), \( p1 = x[1] + y \) in \( (p0, p1) \)
let x = (1, 2)

, y = let tmp = (10, 20)
  in tmp[0] + tmp[1]

, p0 = x[0] + y
, p1 = x[1] + y

in

(p0, p1)
let \( \mathbf{x} = (1, 2) \), \( \mathbf{y} = \) let \( \mathbf{tmp} = (10, 20) \)
    in \( \mathbf{tmp}[0] + \mathbf{tmp}[1] \)
    , \( \mathbf{p0} = \mathbf{x}[0] + \mathbf{y} \)
    , \( \mathbf{p1} = \mathbf{x}[1] + \mathbf{y} \)
  in \( (\mathbf{p0}, \mathbf{p1}) \)
let x = (1, 2)
, y = let tmp = (10, 20)
  in tmp[0] + tmp[1]
, p0 = x[0] + y
, p1 = x[1] + y
in
(p0, p1)
let x = (1, 2)
 , y = let tmp = (10, 20)
   in tmp[0] + tmp[1]
 , p0 = x[0] + y
 , p1 = x[1] + y
 in (p0, p1)

ex1: garbage at end
let $x = (1, 2)$

, $y = \text{let } \text{tmp} = (10, 20) \text{ in } \text{tmp}[0] + \text{tmp}[1]

, $p0 = x[0] + y$

, $p1 = x[1] + y$

in

(in)

(p0, p1)

Result (rax) = 0x21

ex1: garbage at end
let x = (1, 2)
, y = let tmp = (10, 20)
  in tmp[0] + tmp[1]
, p0 = x[0] + y
, p1 = x[1] + y
in
(p0, p1)

Suppose we had a smaller, 4-word heap

ex1: garbage at end
let x = (1, 2)
  , y = let tmp = (10, 20)
       in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in (p0, p1)
let \( x = (1, 2) \), \( y = \) let \( \text{tmp} = (10, 20) \) in \( \text{tmp}[0] + \text{tmp}[1] \), \( p0 = x[0] + y \), \( p1 = x[1] + y \) in \( (p0, p1) \)

Out of memory!
Can’t allocate \((p0, p1)\)
let \( x = (1, 2) \), \( y = \) let \( \text{tmp} = (10, 20) \) in \( \text{tmp}[0] + \text{tmp}[1] \), \( p0 = x[0] + y \), \( p1 = x[1] + y \) in \( (p0, p1) \)

\((10, 20)\) is “garbage”

Q: How to determine if cell is garbage?
```plaintext
let x = (1, 2), y = let tmp = (10, 20) in tmp[0] + tmp[1], p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

(10, 20) is "garbage"
```
let \( x = (1, 2) \), \( y = \) let \( \text{tmp} = (10, 20) \) in \( \text{tmp}[0] + \text{tmp}[1] \), \( p0 = x[0] + y \), \( p1 = x[1] + y \) in \( (p0, p1) \)
let x = (1, 2)
  , y = let tmp = (10, 20)
    in tmp[0] + tmp[1]
  , p0 = x[0] + y
  , p1 = x[1] + y
in
(p0, p1)

Result (rax) = 0x11
Garter / GC

Example 2
ex2: garbage in the middle

```
let y = let tmp = (10, 20)
    in tmp[0] + tmp[1]
  , x = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
in (p0, p1)
```

Start with a 4-word heap
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

ex2: garbage in the middle
let y = let tmp = (10, 20)
    in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in
(p0, p1)
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
   , x = (1, 2)
   , p0 = x[0] + y
   , p1 = x[1] + y
in (p0, p1)
```ml
let y = let tmp = (10, 20)
  in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)
```

ex2: garbage in the middle
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

Out of memory!
Can’t allocate (p0, p1)
let y = let tmp = (10, 20)
    in tmp[0] + tmp[1]
  , x = (1, 2)
  , p0 = x[0] + y
  , p1 = x[1] + y
  in (p0, p1)

Let's reclaim & recycle garbage!
let y = let tmp = (10, 20) 
    in tmp[0] + tmp[1] 
, x = (1, 2) 
, p0 = x[0] + y 
, p1 = x[1] + y 

in (p0, p1)

ex2: garbage in the middle

Let's reclaim & recycle garbage!

QUIZ: Which cells are garbage?
(A) 0x00, 0x08 (B) 0x08, 0x10 (C) 0x18, 0x20 (D) None (E) All
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

ex2: garbage in the middle

QUIZ: Which cells are garbage?
Those that are not reachable from stack
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in
(p0, p1)

ex2: garbage in the middle

QUIZ: Which cells are garbage?
Those that are not reachable from stack
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

Q: How to reclaim space? Why is it not enough to rewind r15?
let y = let tmp = (10, 20)
  in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)

ex2: garbage in the middle

Lets reclaim & recycle garbage!

Why is it not enough to rewind r15?
Want free space to be contiguous (i.e. go to end of heap)
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

Solution: Compaction

Copy “live” cells into “garbage” ...
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

ex2: garbage in the middle

Solution: Compaction
Copy “live” cells into “garbage” …
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)

Let's reclaim & recycle garbage!

Solution: Compaction

Copy "live" cells into "garbage" ...
let y = let tmp = (10, 20)
  in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)

ex2: garbage in the middle

Solution: Compaction

Copy “live” cells into “garbage” …
let y = let tmp = (10, 20) in tmp[0] + tmp[1], x = (1, 2), p0 = x[0] + y, p1 = x[1] + y in (p0, p1)

Let's reclaim & recycle garbage!

Solution: Compaction

Copy “live” cells into “garbage” … and then … rewind r15!
let y = let tmp = (10, 20)
    in tmp[0] + tmp[1]

, x = (1, 2)

, p0 = x[0] + y

, p1 = x[1] + y

in (p0, p1)

Yay! Have space for (p0, p1)
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in
(p0, p1)

Yay! Have space for (p0, p1)
let y = let tmp = (10, 20)
in tmp[0] + tmp[1]
, x = (1, 2)
, p0 = x[0] + y
, p1 = x[1] + y
in (p0, p1)

Result (rax) = 0x09
Garter / GC
Example 3
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
    x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
    x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
  x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
    , x = (y, y + 1)
    , z = foo(100, 200)
in
    x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + y + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
    x[0] + y + z

Return (rax) = 30
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Return (rax) = 30
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20), x = (y, y + 1), z = foo(100, 200)
in
    x[0] + z
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
dx[0] + z
```
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
    x[0] + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
    x[0] + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

1 local var (tmp)
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
x[0] + z

Lets reclaim & recycle garbage!
def foo(p, q):
    let tmp = (p, q)
in tmp[0] + tmp[1]

let y = foo(10, 20), x = (y, y + 1), z = foo(100, 200)
in x[0] + z

Let's reclaim & recycle garbage!

QUIZ: Which cells are garbage?

(A) 0x00, 0x08 (B) 0x08, 0x10 (C) 0x10, 0x18 (D) None (E) All
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Lets reclaim & recycle garbage!

QUIZ: Which cells are garbage?
Those that are not reachable from any stack frame
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in x[0] + z
```

Let's reclaim & recycle garbage!

**QUIZ:** Which cells are garbage?

Those that are *not reachable from any stack frame*
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Let's reclaim & recycle garbage!

Which cells are garbage?
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
x[0] + z

Compact the live cells
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
x[0] + z

Compact the live cells
def foo(p, q):
  let tmp = (p, q)
  in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
  x[0] + z

Compact the live cells
```python
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
```

Compact the live cells ... then rewind r15
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Compact the live cells ... then rewind r15
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Problem???
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
    x[0] + z
```

Problem! Have to REDIRECT existing pointers
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
x[0] + z

1. Compute **FORWARD** addrs
(i.e. new compacted addrs)
ex3: garbage in the middle (with stack)

```
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
```

1. Compute **FORWARD** addrs
e.g. 0x11 —> 0x01
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
    x[0] + z

1. Compute **FORWARD** addrs
e.g. \texttt{0x11} $\rightarrow$ \texttt{0x01}

2. **REDIRECT** addrs on stack
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

1. Compute **FORWARD** addrs e.g. 0x11 —> 0x01
2. **REDIRECT** addrs on stack
3. **COMPACT** cells on heap
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
```

1. Compute **FORWARD** addrs e.g. `0x11 —> 0x01`

2. **REDIRECT** addrs on stack

3. **COMPACT** cells on heap
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Yay! Have space for (p, q)
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in
    x[0] + z

Yay! Have space for (p, q)
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z

Return (rax) = 300
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)

in x[0] + z
```

Return (rax) = 300
def foo(p, q):
    let tmp = (p, q)
    in tmp[0] + tmp[1]

let y  = foo(10, 20)
 , x  = (y, y + 1)
 , z  = foo(100, 200)
in
    x[0] + z
ex3: garbage in the middle (with stack)

```python
def foo(p, q):
    let tmp = (p, q)
in tmp[0] + tmp[1]

let y = foo(10, 20)
, x = (y, y + 1)
, z = foo(100, 200)
in
x[0] + z
```

Return (rax) = 30+300 = 330
Garter / GC
Example 4
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
        in sum(l1)
    , l = range(t1, t1 + 3)
in
    (1000, l)

**call** range(0, 3)
```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

QUIZ: What is heap when range(0,3) returns?

(A) 0 0x11 1 0x21 2 false

(B) 2 false 1 0x01 0 0x11
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

Result sum(0x11) = 3
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, t = range(t1, t1 + 3)
in
(1000, l)
```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)
```

**call** range(3,6)
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
  let l1 = range(0, 3)
  in sum(l1)
,
  l = range(t1, t1 + 3)
in (1000, l)

call range(3,6)

QUIZ: What is the value of l?
(A) 0x30 (B) 0x31 (C) 0x50 (D) 0x51 (E) 0x60
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

Yikes! Out of Memory!
ex4: recursive data

QUIZ: Which cells are “live” on the heap?

(A) 0x00
(B) 0x10
(C) 0x20
(D) 0x30
(E) 0x40
(F) 0x50
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

1. **MARK** live addrs
2. Compute **FORWARD** addrs
3. **REDIRECT** addrs on stack
4. **COMPACT** cells on heap
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

1. MARK live addr
reachable from stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

1. MARK live addr
reachable from stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

1. MARK live addr
reachable from stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

1. **MARK** live addr
    reachable from stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)

, l = range(t1, t1 + 3)

in
(1000, l)

1. MARK live addr
reachable from stack
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)
```

1. **MARK** live addrs
   reachable from stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

Done!
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

2. Compute FORWARD addrs
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)
```

2. Compute FORWARD addrs
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)
```

2. Compute FORWARD addr
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

2. Compute FORWARD addr
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

ex4: recursive data

2. Compute FORWARD addr
```python
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)
```

2. Compute **FORWARD** addr

---

**ex4: recursive data**

<table>
<thead>
<tr>
<th>addr</th>
<th>0x00 0x08 0x10 0x18 0x20 0x28 0x30 0x38 0x40 0x48 0x50 0x58 0x60</th>
</tr>
</thead>
<tbody>
<tr>
<td>rsp</td>
<td></td>
</tr>
<tr>
<td>rbp</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>0x51</td>
</tr>
<tr>
<td>r15</td>
<td></td>
</tr>
</tbody>
</table>

---

**fwd**

---

**orig**
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

2. Compute FORWARD addr

ex4: recursive data
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
        in sum(l1)
  , l = range(t1, t1 + 3)
in (1000, l)
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

2. Compute FORWARD addrs
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)

, l = range(t1, t1 + 3)
in
(1000, l)
```

2. Compute FORWARD addr
def range(i, j):
    if (j <= i): false else: (i,range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)

2. Compute **FORWARD** addrs
Where should we store the forward addrs?
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

3. REDIRECT addr s on stack
```
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)

(1000, l)

3. REDIRECT addrs on stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

3. REDIRECT addr on stack
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
        in sum(l1)
    , l = range(t1, t1 + 3)
in
    (1000, l)

3. REDIRECT addrs on stack and heap!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
        in sum(l1)
    , l = range(t1, t1 + 3)
in
(1000, l)

3. REDIRECT addrs on stack and heap!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)

in
(1000, l)

3. REDIRECT addrs on stack and heap!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)

in (1000, l)

3. REDIRECT addrs on stack and heap!
def range(i, j):
    if \( j \leq i \): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

3. REDIRECT addrs on stack and heap!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
       in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

4. COMPACT cells on heap
Copy cell to forward addr!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

4. COMPACT cells on heap
Copy cell to forward addr!
```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)
```

4. COMPACT cells on heap
Copy cell to forward addr!
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
    , l = range(t1, t1 + 3)
in (1000, l)
```

4. **COMPACT** cells on heap
   Copy cell to forward addr!
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in (1000, l)
```

4. **COMPACT** cells on heap

Copy cell to forward addr!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3) in
(1000, l)

4. COMPACT cells on heap
Copy cell to forward addr!
ex4: recursive data

```python
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)
```

4. **COMPACT** cells on heap
Copy cell to forward addr!
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)
in
(1000, l)

GC Complete!
Have space for (1000, l)
def range(i, j):
    if (j <= i): false else: (i, range(i+1, j))

def sum(l):
    if l == false: 0 else: l[0] + sum(l[1])

let t1 =
    let l1 = range(0, 3)
    in sum(l1)
, l = range(t1, t1 + 3)

in
(1000, l)

GC Complete!
Have space for (1000, l)