Garter
Garbage Collection

Ranjit Jhala | UCSD
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 tmp = (10, 20)
in tmp[0] + tmp[1], p0 = x[0] + y, p1 = x[1] + y
in (p0, p1)

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

Result (rax) = \( 0x21 \)
```haskell
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  
  \[ x = (1, 2), \quad y = \begin{align*} 
  & \text{let} \quad \text{tmp} = (10, 20) \\
  & \text{in} \quad \text{tmp}[0] + \text{tmp}[1] \\
  \end{align*} \]
  \[ \text{in} \quad \text{tmp}[0] + \text{tmp}[1] \]
  \[ \quad \text{p0} = x[0] + y \]
  \[ \quad \text{p1} = x[1] + y \]
  \[ \quad \text{in} \]
  \[ (\text{p0}, \text{p1}) \]
ex1: garbage at end

```haskell
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)
```

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

\((10, 20)\) is "garbage"

Q: How to determine if cell is garbage?
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 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 \( \text{tmp}[0] + \text{tmp}[1] \), \( p_0 = x[0] + y \), \( p_1 = x[1] + y \) in \( (p_0, p_1) \)

**Result** \( (\text{rax}) = 0x11 \)

ex1: garbage at end
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)
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]
    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)
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)

Lets 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

QUIZ: Which cells are garbage?
(A) 0x00, 0x08 (B) 0x08, 0x10 (C) 0x18, 0x20 (D) None (E) All
ex2: garbage in the middle

```ocaml
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!

QUIZ: Which cells are garbage?

Those that are not reachable from stack
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)
```

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?
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)
```

Why is it not enough to rewind r15?
Want free space to be contiguous (i.e. go to end of heap)
let \( y = \) let \( \text{tmp} = (10, 20) \in \text{tmp}[0] + \text{tmp}[1] \)
\, \( x = (1, 2) \)
\, \( p_0 = x[0] + y \)
\, \( p_1 = x[1] + y \)
\in (p_0, p_1)

**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)

Solution: Compaction
Copy “live” cells into “garbage” ...
ex2: garbage in the middle

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

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)

ex2: garbage in the middle

Solution: Compaction

Copy “live” cells into “garbage” ... and then ... rewind r15!
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)
```

Yay! Have space for (p0, p1)
ex2: garbage in the middle

```haskell
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

ex2: garbage in the middle
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
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] + 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
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
```
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
```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
```
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

Let's reclaim & recycle garbage!
Let's reclaim & recycle garbage!

**QUIZ: Which cells are garbage?**

(A) 0x00, 0x08    (B) 0x08, 0x10    (C) 0x10, 0x18    (D) None    (E) All
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
```

Let's reclaim & recycle garbage!

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x08</td>
<td>0x10</td>
<td>0x18</td>
<td>0x20</td>
</tr>
</tbody>
</table>

**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!

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!

Which cells are garbage?
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
```

Compact the live cells
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
```

Compact the live cells
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
```

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 ... 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
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???
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
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)

let x[0] + z
```

1. Compute **FORWARD** addrs (i.e. new compacted addrs)
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** addr
   e.g. 0x11 -> 0x01
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. \(0x11 \rightarrow 0x01\)
2. **REDIRECT** addrs on stack

![Diagram with stack frame and memory addresses]
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

1. Compute FORWARD addr
   e.g. 0x11 —> 0x01

2. REDIRECT addr 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)
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

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

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

(A)
\[
\begin{array}{cccccc}
0 & 0x11 & 1 & 0x21 & 2 & false
\end{array}
\]

(B)
\[
\begin{array}{cccccc}
2 & false & 1 & 0x01 & 0 & 0x11
\end{array}
\]
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

r15
2 false 1 0x01 0 0x11
0x00 0x08 0x10 0x18 0x20 0x28 0x30 0x38 0x40 0x48 0x50 0x58 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)
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)
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)

(1000, l)

QUIZ: What is the value of l?
(A) 0x30 (B) 0x31 (C) 0x50 (D) 0x51 (E) 0x60
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)

Yikes! Out of Memory!
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

ex4: recursive data
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)

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)

1. **MARK** live addrs
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)

1. **MARK live addrs**
reachable from stack
```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
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
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
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)

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
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)

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)

(1000, l)

2. Compute FORWARD addrs
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
ex4: recursive data

2. Compute **FORWARD** addrs
Where should we store the forward addrs?
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
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 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!
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 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!
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 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 addr on stack and heap!
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!
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!
```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)

ex4: recursive data

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!
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)
QUIZ: What should `print(0x21)` show?

(A) `(0, (1, (2, false)))`
(B) `(3, (4, (5, false)))`
(C) `(0, (1, (2, (3, (4, (5, false))))))`
(D) `(3, (4, (5, (0, (1, (2, false))))))`
(E) `(2, (1, (0, (3, (4, (5, false))))))`
ex4: recursive data

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

(A) 0x00
(B) 0x10
(C) 0x20
(D) 0x30
(E) 0x40
(F) 0x50