FOX
Garbage Collection
FOX / 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) 

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

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 $x = (1, 2)$,
    $y = \text{let } \text{tmp} = (10, 20) \text{ in } \text{tmp}[0] + \text{tmp}[1]$
    $p0 = x[0] + y$
    $p1 = x[1] + y$
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 (eax) = 0x11

ex1: garbage at end
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 \( (p0, p1) \)

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 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)

1  2  10  20
0x00 0x04 0x08 0x0c 0x10
ebp
esi
32 p1
31 p0
30 y
0x01 x
esi
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)
```

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

Result (eax) = 0x09
FOX / GC
Example 2
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)
ex2: garbage in the middle

```plaintext
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]
, 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)

Lets reclaim & recycle garbage!
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?
(A) 0x00, 0x04 (B) 0x04, 0x08 (C) 0x08, 0x0c (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)

Let's reclaim & recycle garbage!

Ex2: garbage in the middle

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)
```

Let's reclaim & recycle garbage!

**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 esi?
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's reclaim & recycle garbage!

Why is it not enough to rewind **esi**?

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)

Lets reclaim & recycle garbage!

Solution: Compaction

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

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)

Let’s reclaim & recycle garbage!

Solution: Compaction

Copy “live” cells into “garbage” …
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)
```

**Solution: Compaction**

Copy “live” cells into “garbage” ... then rewind esi!
```
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)
ex2: garbage in the middle

```javascript
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 (eax) = 0x09
Example 3
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] + y + z
```

3 local vars x, 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)

```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] + 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)

```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] + y + z

Return (eax) = 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
```

Return (eax) = 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
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
```
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
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

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

Lets reclaim & recycle garbage!

QUIZ: Which cells are garbage?

(A) 0x00, 0x04  (B) 0x04, 0x08  (C) 0x08, 0x0c  (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
```

Traverse Stack from top (esp) to bottom (ebp0) to mark reachable cells.

Let's reclaim & recycle garbage!

<table>
<thead>
<tr>
<th>10</th>
<th>20</th>
<th>30</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x04</td>
<td>0x08</td>
<td>0x0c</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!

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
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
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 esi
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 esi
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)
in
x[0] + z
```

1. Compute **FORWARD** addrs
   (i.e. new compacted addrs)
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** addr
   e.g. \texttt{0x09} \rightarrow \texttt{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. 0x09 → 0x01

2. **REDIRECT** addrs on stack
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. \texttt{0x09} $\rightarrow$ \texttt{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. 0x09 —> 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)
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 \( (eax) = 300 \)
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 (eax) = 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 (eax) = 30+300 = 330
```
```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)
```
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)  

\[
\begin{array}{cccccccc}
0 & 0x09 & 1 & 0x11 & 2 & \text{false} & \hline
0x00 & 0x04 & 0x08 & 0x0c & 0x10 & 0x14 & 0x18 & 0x1c & 0x20 & 0x24 & 0x28 & 0x2c & 0x30
\end{array}
\]

(B)  

\[
\begin{array}{cccccccc}
2 & \text{false} & 1 & 0x01 & 0 & 0x09 & \hline
0x00 & 0x04 & 0x08 & 0x0c & 0x10 & 0x14 & 0x18 & 0x1c & 0x20 & 0x24 & 0x28 & 0x2c & 0x30
\end{array}
\]
```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)
```

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)

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)
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)

esi

2 false 1 0x01 0 0x09
0x00 0x04 0x08 0x0c 0x10 0x14 0x18 0x1c 0x20 0x24 0x28 0x2c 0x30
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)
call range(3,6)
```

QUIZ: What is the value of `l`?
(A) 0x18 (B) 0x19 (C) 0x28 (D) 0x29 (E) 0x30
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)
```

Yikes! Out of Memory!
QUIZ: Which cells are “live” on the heap?

(A) 0x00
(B) 0x08
(C) 0x10
(D) 0x18
(E) 0x20
(F) 0x28

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)

1. **MARK** live addrs
2. Compute **FORWARD** addrs
3. **REDIRECT** addrs on stack
4. **COMPACT** cells on 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)
```

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)

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

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)
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)
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
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

```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
Where should we store the 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)

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
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 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 addrss 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 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)

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!
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)

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)
QUIZ: What should `print(0x11)` 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))))))
QUIZ: Which cells are “live” on the heap?

(A) 0x00
(B) 0x08
(C) 0x10
(D) 0x18
(E) 0x20
(F) 0x28